AN EVALUATION AND COMPARISON OF THREE NONLINEAR PROGRAMMING CODES

Ralph John Waterman

THESIS

AN EVALUATION AND COMPARISON OF
THREE NONLINEAR PROGRAMMING CODES

by

Ralph John Waterman

Thesis Advisor: G. H. Bradley

March 1976

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) An Evaluation and Comparison of Three Nonlinear Programming Codes		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis March 1976 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Ralph John Waterman		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		12. REPORT DATE March 1976 13. NUMBER OF PAGES 94
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
nonlinear programming
computational testing
optimization

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This study evaluates and compares the production use of three nonlinear programming codes. The three codes and their developers are: SUMT by W. C. Mylander, R. L. Holmes and G. P. McCormick, GRG by L. S. Lasdon, A. D. Waren, M. W. Ratner and A. Jain, and GRAVES by G. W. Graves. This is the first computer comparison of these three particular codes. Each code was evaluated with respect to the time and sophistication required of the user and the degree of mandatory or potential interaction between the code and the

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |
(Page 1)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. analyst. The comparison criteria were accuracy, robustness, efficiency and ease of utilization.

Eight current and realistic test problems employing from 9-100 variables and 2-20 constraints were used.

The results revealed that no single code was superior or inferior in all aspects. The choice of an optimal code among these three would be dependent upon the problems to be solved, the ability of the analyst and the desire of the analyst to alter the code for his own purposes.

An Evaluation and Comparison of Three Nonlinear Programming Codes

by

Ralph John Waterman
Lieutenant, United States Navy
B.A., Columbia University, 1969

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
March 1976

ABSTRACT

This study evaluates and compares the production use of three nonlinear programming codes. The three codes and their developers are: SUMT by W.C. Mylander, R.L. Holmes and G.P. McCormick, GRG by L.S. Lasdon, A.D. Waren, M.W. Ratner and A. Jain, and GRAVES by G.W. Graves. This is the first computer comparison of these three particular codes. Each code was evaluated with respect to the time and sophistication required of the user and the degree of mandatory or potential interaction between the code and the analyst. The comparison criteria were accuracy, robustness, efficiency and ease of utilization.

Eight current and realistic test problems employing from 9-100 variables and 2-20 constraints were used.

The results revealed that no single code was superior or inferior in all aspects. The choice of an optimal code among these three would be dependent upon the problems to be solved, the ability of the analyst and the desire of the analyst to alter the code for his own purposes.

TABLE OF CONTENTS

I. INTRODUCTION

This study compares and evaluates the production use of three nonlinear programming codes. The analyst was not an author of any of the codes, had some prior programming experience in a standard classroom context but had no previous exposure to a production nonlinear programming code or numerical solution of nonlinear programming problems.

Obtaining a solution to a nonlinear problem of more than a few variables requires the utilization of two components: an algorithm coded for a computer and an analyst with an understanding of the problem for which a solution is desired. Finding the solution for a nonlinear problem requires an interaction between the man and the code-a complex interaction with subtle relationships between each component. Every problem will require a different degree of expertise from both the analyst and the code and it is precisely this variation between problems that prevents the separation of the analyst's responsibilities from those of the nonlinear programming code. Excellent nonlinear programming production codes provide a multitude of approaches, but the analyst is the one controlling the avenue selected. A production type code has to be general. The code must be able to respond to the variation in problem characteristics: in variables, many or few, objective function, linear or highly nonlinear, constraints, many, few, linear, nonlinear, equality or inequality. An adequate code is able to handle all of these possibilities with some degree of success. In many instances however, the degree of success is dependent upon the ability of the analyst and his familiarity with the internal actions and reactions of the code and its mathematical approach.

A user of a nonlinear programming code must be able to

communicate with the code, he must be able to provide the code with a useable interpretation of the problem and he must be able to interpret the results provided by the code. This requires the ability to program the problem in the computer language employed by the code, the ability to compute derivatives, a knowledge of the correct parameters to select for the code and, even more important, enough of an understanding of the printout to determine if a global minimum has been obtained and if not what changes might remedy the problem. A nonlinear programming code is a black box requiring quality input in order to provide quality output. The most sophisticated code is of little value without a capable user.

A practical evaluation of any nonlinear programming code must first examine the analyst-code interaction, an interaction loaded with frustrations. All codes require subroutines provided by the user to evaluate the objective function and constraint functions. Most codes also need first derivatives, and some require second derivatives, in order to determine directions for improved solutions. The code has no knowledge of the functions other than what is available from these subroutines. A general evaluation criterion is the relative ease with which the user can formulate these subroutines, especially the first time user. The code cannot operate without these function and gradient routines; the accuracy and correctness of these function evaluations determines the code's ability to interact with the user.

There always exists the element of human error in coding these subroutines, errors which if undetected will result in inaccurate solutions. A code which leaves the user with no clue as to the type and location of such errors will cause delays in debugging. The total cost of reaching a solution is not solely a function of the computer time

used but is jointly dependent on the time of the man and machine. This author's experience indicates that, while an analyst does over a period of time develop an increased efficiency in using a code, for large difficult nonlinear problems the cost of the analyst's time will always be a significant part and often will be the largest single component of the total cost.

This study is also a comparison of the codes, hopefully a guide to potential users as to which of the three nonlinear prcgramming codes might best satisfy a particular need. Numerous nonlinear programming codes have been developed over the years without a similar degree of interest being shown in testing and comparing their specific convergence abilities. The three codes chosen will be referred to as GRG, SUMT, and GRAVES. The GRAVES code was formulated in 1964 by Professor G. W. Graves and was utilized in March 1974 to obtain a solution to a sortie allocation nonlinear programming model prepared for the Department cf Defense Program Analysis and Evaluation. The GRG code, a Generalized Reduced Gradient Algorithm, was developed jcintly by L.S. Lasdon, A.D. Waren, M.W. Ratner and A. Jain as members of the Computer and Information Science Department, Cleveland State University and the Department of Operations Research, Case Western Reserve University. The SUMT code was developed by W. Charles Mylander, R.I. Holmes and G. P. McCormick as members of the Research Analysis Corporation, McLean, Virginia.

This is the first computer comparison of these three particular codes. The GRAVES and GRG codes do not, at this time, have widespread usage and (before this study) have been primarily utilized by the authors and their close associates. For easier evaluation of the results the SUMT version 4 NLP code was chosen as the third and final code because of its more widely known qualities and limitations.

This study was to compare the codes from the standpoint of a user with only limited knowledge of the internal operations of each. Although studies and tests performed by someone thoroughly familar with the vagarities and intrinsic behavior of a code do reveal the upper bounds of a code's capabilities, it was desired to test the accuracy and ease of utilization when each code was used by an analyst in a normal production atmosphere. If the input routines and tuning tolerances were originated by a user other than each code's author would they continue to perform satisfactorily?

The codes were compared on the ease of preparation for the first time user and on their response to an analyst's effort to interact with the internal operations of the code. After developing a feel for the weaker characteristics of a code are alterations and improvements possible? The generality of large production codes is such that an innocent alteration in one area of the code may cause complete disruption elsewhere. Perhaps all-purpose codes are not really desirable for an analyst who over a period of years may wish to alter an algorithm to suit his specific needs. These three codes have very different approaches to the need for interaction between the code and the user and in comparing the user-code communication for each of the three codes one must keep in mind the fact that the needs and talents of each analyst vary as will the code best suited to these needs.

These routines are coded in the same programming language (FORTRAN) and run on the same machine, the IBM 360/67. This eliminated the necessity of attempting to compare run times from two or more different machines. The IBM 360/67 is a multiprogramming machine; there may be a variation in run times of identical jobs of as much as 25% depending on the loading of other work on the machine at the

time of execution. There is no effective way to deal with this problem. All the codes were run under the same conditions.

The best known study to date is a paper [Ref. 1] presented by Colville in 1967, comparing 28 different codes which were grouped according to four general classifications:

1. Direct search methods

2. Small step gradient methods

3. Large step gradient methods

4. Second derivative methods

Colville chose eight problems from among a large selection made available to him by the participants. Each participant was then asked to solve this set of eight problems using his own NLP code and the computer facility of his choice. The results of Colville's study revealed many difficulties, inaccuracies and discrepancies inherent in any cooperative effort of this size. Of course, the time and effort required for one person to program and test 28 NLP codes would be prohibitive. Colville concluded that the efficiency and performance of a nonlinear programming code can be greatly affected by the method and efficiency of implementation on a computer. Several codes utilizing identical mathematical approaches had a large variance in accuracy and time. Algorithms which were identical from the theoretical viewpoint were not at all comparable in reliability, accuracy and efficiency because of considerable differences in the method of programming these algorithms for a computer.

In attempting to standardize the solution times from different computers Colville developed a FORTRAN standard timing program which simply inverts a 40 x 40 matrix ten times. However, the comparison of standardized times is not a precise measure and the Colville study revealed quite a discrepancy between the standardized times of different computers when using the same programming code to solve identical problems.

In any comparative analysis it is first necessary to develop a measure of effectiveness. The criteria necessary for evaluating a nonlinear programming code are complex and there exists no one formula which accurately weighs the various advantages and disadvantages of each code. For instance, the number of function evaluations is the total number of times it is necessary to evaluate the objective and constraint functions, in addition to any first and second derivative evaluations, before reaching the final solution of the problem. Although some studies have used the number of function evaluations required to reach the optimal solution as a measure of the code's efficiency, this is not a very useful criterion for constrained problems. The time required to determine the next point for function evaluation will vary greatly among codes. Thus, the time allotted for function evaluations may be quite insignificant relative to the total solution time and the scalar used to denote the number of function evaluations is really not a fair factor for comparing the efficiency of different nonlinear programming codes. On the other hand, actual computer computation time seemed to be the soundest and simplest measure of efficiency of the codes. However, the quality of solutions can be as important as their cost in terms of computation time.

A. Comparison Criteria

The comparison of the three codes will be based on the following criteria.

1. Accuracy of the final solutions

In evaluating general purpose codes a primary criterion must be whether or not the algorithm can solve the problems presented to an acceptable degree of precision.

2. Robustness of a code

Although any solution which locates a local minimum must be considered somewhat successful, a code which repeatedly attains global minima from both feasible and nonfeasible initial points is certainly more desirable.

3. Speed of solution

Total solution time is a function of the degree of precision desired in the values of the objective functions, the independent variables and the constraints. Because the degree of accuracy depends on the termination criteria employed to stop the computation, an attempt was made in this study to vary the termination parameters so that each code attained similar degrees of accuracy in all functions. This procedure was not always successful for each different test problem because of the variety of termination criteria for each of the codes. The stopping criterion may be a function of the numerical components of the gradient, complete Kuhn-Tucker satisfaction, or a minimum absolute change may be required in the objective function between iterations. A steep slope or a relatively flat minimum can cause each code to attain a different level of precision even though the actual user supplied tolerances may be identical.

Not all codes are designed to obtain the same degree of precision. A fast, good solution may be needed or a solution with extreme accuracy may be desired. The code's design determines the ability to react to these different precision requirements and requiring precision beyond reasonable limits will completely distort time comparisons between codes. Thus, in order to achieve comparable times for each code in terms of accuracy, parameters were altered after successful solutions had been attained until similar but not identical degrees of accuracy were attained.

Perhaps the most accurate timing comparison between the codes would be net computation time required for each problem since net time does not include the time for processing read and write statements and delays caused by other unrelated machine activity. However, these additional times do affect the actual turnaround time, an important factor for any analyst working against a deadline. As the study progressed it was discovered that the solution times were typically significantly different for most problems and therefore total CPU time was chosen as an adequate criterion for the comparison data.

4. Ease of setting up the user supplied subroutines for the evaluation of the objective function and the constraint equations

In order to utilize any nonlinear programming code it is necessary for the user to supply one or more subroutines. Every code requires a routine for evaluating the objective and constraint functions and many require the first derivatives, while a few even need the second derivatives. The physical size and degree of difficulty of the setup subroutines that must be prepared for code utilization are two very important factors. Difficult and lengthy codes are very vulnerable to human error. The

coding and debugging of first and second analytical derivatives (especially problems without very sparce, or recurring special structure in the constraints such as block diagonal, angular, staircase, or other form) can be a trying procedure. A significant factor is the value of the user's preparation time. In some circumstances a less efficient algorithm that can be easily coded for use is more practical than an efficient code requiring many man hours for setup. A comparison along these lines is highly subjective and will vary from situation to situation but the relative degree of difficulty involved in using each code must certainly be considered.

5. Output available to aid in debugging new programs

Improperly coded functions and derivatives are bound to occur for almost every problem. It is important for a code to provide appropriate output to aid in locating the particular mis-coded equations which are preventing the attainment of the solution. Most nonlinear programming codes do have the capability of providing output designed for checking for consistency between the function and gradient evaluations.

6. Computer memory region required

At most computer centers main memory requirements largely determine the priority of job requests and as such codes requiring large amounts of core will have much longer turnaround times. Also many commercial data centers charge for computer service on a kilobyte-second basis making region equally important with time of computation.

7. Failure mode

All codes fail at some time or another, usually

for one of two reasons: either they are not suited for the particular type of problem involved and a solution can not be reached, or one of the user supplied parameters or perhaps even the initial starting point needs to be altered. Some codes fail "hard" and the user cannot determine the cause from the available output while others fail "soft" with pinpoint details as to the last available point, function and gradient values and, in the case of a binding tolerance, the particular parameter that needs to be altered. Quick and accurate determination of the cause of failure can save an analyst many trial and error computer runs. This is especially valuable in the case of long turnaround times.

8. Growth possibilities

Contemporary nonlinear problems of 100 variables or less are really small or moderate sized and the prospect of expanding a code for hundreds and even thousands of variables is considered. The growth in code and CPU time is certainly not linear and a code's basic structure may severely curtail its feasibility for expansion.

B. Test Problems

For this study eight problems were selected encompassing various degrees of difficulty. The number of variables and constraints range from 9 to 100 and 2 to 20 respectively. Most of the problems contain combinations of linear, nonlinear, equality and inequality constraints. Six of the problems have been previously published with the best known solutions.

In testing any nonlinear programming code a prior knowledge of the global minimum is certainly helpful in evaluating the precision of the code. The last two problems

were expansions of models with fewer variables and each code was tested against the smaller problem with only changes in the initial parameters necessary for the actual test problems.

All three codes are sensitive to problem scaling, therefore, no problems with significant (or contrived) magnitudes of difference between the independent variables were utilized. Initially it was intended to test the codes with hundreds of variables but this problem dimensionality was not achieved because of computer memory and time limitations. One problem, developed by the U.S. Army Concepts Analysis Agency, consisting of 500 variables and 45 constraints was dropped because, although the Graves code could solve it using 200K bytes of main memory, the SUMT and GRG codes would require well over 1500K.

II. DESCRIPTION OF CODES

A. SUMT

The SUMT computer code was developed by the Research Analysis Corporation in the 1960's and improved throughout the decade. SUMT implements the Sequential Unconstrained Minimization Technique for nonlinear programming and other computation techniques developed in Chapter 8 of the book by Fiacco and McCormick [Ref. 2]. SUMT is coded in FORTRAN IV and employs double precision arithmetic. The mixed interior-exterior penalty function is used and several options are available for minimizing the penalty function. Specifically the SUMT code addresses the question of locating the N dimensional vector $x(*)$ that solves

$$\text{minimize} \quad f(x)$$

$$\text{subject to} \quad g_j(x) \geq 0, \quad j = 1,2,\ldots,m$$

$$h_j(x) = 0, \quad j = m+1,\ldots,m+p$$

The SUMT algorithm has been developed to solve the nonlinear programming problem in which the objective function and the inequality constraints can be nonlinear functions of the variables but in order to guarantee convergence the equality constraints should be linear functions of the independent variables. However, very accurate SUMT solutions have been obtained in many problems with nonlinear equality constraints.

The SUMT approach is to solve repetitively a sequence of unconstrained problems whose solutions in the limit

approach the minimum of the constrained nonlinear programming problem. To convert problems the SUMT code uses the penalty function

$$P\left(x^{(k)},r^{(k)}\right) = f(x) + \frac{1}{r^{(k)}} \sum_{i=1}^{m} h_i^2\left(x^{(k)}\right) - r^{(k)} \sum_{i=m+1}^{p} \ln g_i\left(x^{(k)}\right)$$

where the factors r are positive and decreasing as the minimization progresses. As r is reduced, the barrier effect is reduced and $x(\bullet)$ may move closer to an inequality constraint boundary. For the equality constraints the sum of squares is used, thus, as r approaches zero the equality constraints are closely satisfied. If an initial feasible point is not provided by the user, repeated application of the SUMT method is used to obtain an $x(\bullet)$ vector that is an interior feasible point. However, in the case of highly nonlinear constraints a great deal of time may be used to initially locate a feasible point. When available the user should provide a feasible starting point.

Utilizing the first feasible point and the initial r value provided by the user $x^{(1)}$ is determined by minimizing $P\left(x^{(0)},r^{(0)}\right)$. Then $r^{(1)}$ is computed and utilized to determine $x^{(2)}$ by minimizing $P\left(x^{(1)},r^{(1)}\right)$ and so forth until a minimum is obtained. Although three possible settings are available, $r^{(0)}=1$ is the most practical. Computation time becomes longer for larger initial values for $r^{(0)}$. Too large an initial r distorts the initial P function and forces $x(\bullet)$ too far into the feasible interior while too small a $r^{(0)}$ starts $x(\bullet)$ too close to the boundary.

Overall, $r^{(0)}=1$ is a good penalty setting.

The direction of search for locating a new minimum of the P function is obtained by Newton's Method. This requires multiplying the inverse of the Hessian matrix by the gradient of the P function. Not only is the time element quite high but the inverse of the Hessian matrix can become ill-conditioned. Thus as the extremum is approached the search directions may become misleading necessitating additional time for the code to backtrack in search of another feasible direction. Once the search direction is established, SUMT uses a Fibonacci search to locate the minimum of the P function.

1. Documentation

The necessary steps for SUMT operation are clearly and thoroughly presented for the first time user. Reference 3 by Mylander, Holmes and McCormick greatly facilitates the use of the SUMT code by the unfamilar programmer. This document concisely presents a brief summary of the theoretical background for the algorithm implemented by SUMT. The step by step logic of the computer routine is developed and accompanied by a flowchart of the minimization process. Reference 3 continues with a general description of the computer program listing and discusses the SUMT internal subroutines. The program is modular in structure to allow for easy changes in options, input, output and tolerances for determining when the optimum solution has been obtained. A table is included which details the relationship between the subroutines with inter-routine communications being handled by labelled COMMON regions. One or more paragraphs are allotted to each subroutine indicating its function and general logic.

A separate chapter discusses the user supplied subroutines. A complete and detailed description of the necessary common regions for parameter passing between the SUMT main minimization program and the user routines is supplied, along with the input and output requirements of the four subroutines.

The parameter cards and their necessary formats are covered including a complete description of the available options and a recommended setting for each of the tolerance parameters. Three dimensional figures are included showing the data deck structure necessary for the SUMT code setup.

Finally the the document presents a complete problem formulation and solution. Starting with the initial equations, illustrations are provided which display the exact input deck setup and the actual computer output with final and intermediate solutions.

2. Implementation

To implement SUMT the user must supply four subroutines: READIN, RESTNT, GRAD1 and MATRIX.

a. READIN

READIN is called only once and allows the user to read in the coefficients necessary for objective and constraint function evaluation. This routine was utilized in this study to initialize an array which tabulated the number of calls to each user supplied subroutine. Of course, any initial printout desired by the user can be handled in READIN also.

b. RESTNT (I,VAL)

If I = 0, RESTNT returns VAL = f(x). For $I \neq 0$, VAL = $g_i(x)$. The current x(•) values are found in the common region.

c. GRAD1 (I)

When I = 0, GRAD1 must place the gradient of the objective function f(x) in the array DEL(•). When $I \neq 0$, DEL(•) must contain the gradient of the constraint corresponding to I.

The SUMT main program contains a subroutine DIFF1 which evaluates the gradient by a numerical differencing routine. DIFF1 can be called by GRAD1 if explicit derivatives are not possible or feasible to compute.

d. MATRIX (J,L)

This routine places the Hessian of the Jth equation in a N x N array where N equals the number of variables in the x(•) vector. SUMT also provides a subroutine DIFF2 which can be used to evaluate the Hessian by numerical differencing.

For data input the user must supply a parameter card, two option cards, a tolerance card and cards containing an initial x(•) vector. The format for these cards are preset in SUMT. The two most critical parameters are EPSI and THETAO. These two tolerances determine, respectively, when an unconstrained minimum has been achieved for each subproblem and if the solution to the NLP problem has been approximated. The trivial constraint that all $x_i \geq 0$ can be set automatically by the option cards and extrapolation through the last two or three minima can be utilized if

desired. Any coefficients necessary for function evaluations must be input here also.

3. Printout

SUMT provides a listing of all input parameters, initial x(•) vector values, initial objective function value and the values of the initial constraint functions. The output is controlled by an option card parameter and can be set at two levels, either providing printout after the solution of every subproblem or after every intermediate point. Final and intermediate x(•) vector values are provided along with the minimized objective function. Lagrange multipliers are printed along with "first-" and "second-order solution estimates." These multipliers and "order" estimates are printed out after each minimum is obtained. These estimates [Ref. 2] are obtained by extrapolating through two and three minima to explore the trajectory of these points. This trajectory accelerates algorithm convergence and provides a close approximation of the local or global minimum early in the minimization progress.

4. Debugging Aids

Any subroutine requiring the evaluation of as many as 100-variable functions and complicated first and second derivatives is very susceptible to hard-to-detect programming errors. SUMT contains a subroutine called CHECKER which provides a listing of all first and second derivatives obtained by the analytical methods of GRAD1 and MATRIX. A printout then immediately follows containing the same derivatives computed by the numerical differencing subroutines DIFF1 and DIFF2. Any large discrepancy between the evaluation techniques would indicate which subroutines and equations should be double checked.

5. Alterations

Two of SUMT's internal subroutines DIFF1 and DIFF2 were altered slightly for this study. Originally the value used for the differencing was a small positive user supplied value of approximately 10^{-3} or 10^{-4}. This constant value proved inefficient in problems with a large range in the magnitude of the independent variables. Four FORTRAN statements were changed in DIFF1 and DIFF2 so that the value used for differencing was equal to 10^{-4} x (the absolute value of the variable $x_i(\bullet)$) or 10^{-4}, whichever larger. This change proved more efficient in Problems 5 and 8 and in comparing the analytical and numerical derivatives in the CHECKER subroutine.

B. GRG

The GRG code was developed jointly at Cleveland State University and Case Western Reserve University by L.S. Lasdon, A.D. Waren, M.W. Ratner and A. Jain. GRG uses the Generalized Reduced Gradient Algorithm to locate the vector $x(\bullet)$ which solves

$$\text{minimize} \quad g_{M+1}(x)$$

$$\text{subject to} \quad g_i(x) = 0, \qquad i = 1,\ldots,NEQ$$

$$0 \le g_i(x) \le UB(N+i), \qquad i = NEQ+1,\ldots,M$$

$$LB(i) \le x_i \le UB(i), \qquad i = 1,\ldots,N$$

The functions g_i are assumed to be continuously differentiable.

The generalized reduced gradient algorithm uses a modified gradient to solve nonlinear objective functions and nonlinear constraints. By using linear or linearized constraints the method defines new variables that are normal to some of the constraints and transforms the gradient to the new basis. The code operates in two phases, first finding an initial feasible point, and second minimizing the user supplied objective $g_{M+1}(x)$. Of course, time is saved if the user supplied starting point is feasible. If the starting point is nonfeasible the GRG code will locate a feasible point by minimizing a phase I objective function, which is the sum of the constraint violations.

Once a feasible point is obtained the algorithm determines which constraints are binding for the current $x(\bullet)$ vector. A constraint is binding if it is within EPNEWT of its upper or lower ranges. EPNEWT is a small positive number supplied by the user with a default value of 10^{-4}. GRG solves for K of the natural variables, where K is the number of binding constraints at that time. These are basic variables and are solved in terms of the remaining natural variables. The binding constraints may then be solved by using the reduced gradient to determine a search direction d which is employed in a series of one dimensional searches whose goal is

$$\underset{\alpha>0}{\text{minimize}} \quad f(x(\bullet) + \alpha d)$$

To guarantee that the direction d is always a direction of descent

$$d^T \nabla f(x) < 0$$

must hold. A variant of Newton's Method is used and if convergence is attained and no constraints are violated a new α value is selected and the one dimensional search process continues.

Once a solution is found the constraints are checked for violations. Then the current objective value is compared to the previous best value and the best values for the variables, constraints, objective function and stepsize are stored. If the Newton Method has not converged the stepsize is cut back until no improvement is noted in the constraints for ten iterations or until the maximum constraint violation is less than EPNEWT.

Newton's Method is used to compute the values of the basic variables for given values of the nonbasic variables and then the search direction is determined by using the inverse of the basis matrix. This requires two J x J matrices where J equals the number of binding constraints. In large problems the main high speed computer memory required for the storage and inversion of these matrices becomes a major issue.

For the initial estimates of the basic variables the user may select quadratic extrapolation or a tangent vector is computed. If the basis is degenerate a separate

subroutine is used to compute the search direction using the tangent vector to construct a feasible direction. The (NxN)/2 matrix required for this basis is another array contributing to the high memory requirements of GRG.

The usual stopping criteria is satisfaction of the Kuhn-Tucker conditions to within EPSTOP, a user supplied tolerance level with a default value of 10^{-3}. The code also employs as stopping criteria limits for the total number of iterations and number of Newton iterations without convergence, or if the relative change in the objective function is less than EPSTOP for three consecutive iterations the present solution is considered to be a minimum.

1. Documentation

The GRG code is documented by two technical memoranda published in November, 1975. "GRG System Documentation" [Ref. 4] and "GRG User's Guide" [Ref. 5] both authored by Lasdon, Waren, Ratner and Jain. Reference 4 briefly describes the version of the Generalized Reduced Gradient Algorithm that is being utilized by the GRG code. A short overview of the entire code is given followed by tables listing all the subroutines with a concise description of each routine's function and its relation to each of the other routines. GRG is coded in FORTRAN IV and uses double precision arithmetic. Inter-routine communication is handled by labeled COMMON regions. Several flowcharts are included which detail the use and derivation of variables and the utilization of parameters and tolerances to influence the logic of the GRG code. Reference 5 provides the first time user with explicit instructions as to the mandatory and optional subroutines which are provided with their necessary input and output. A

complete description of the data cards and the options available for each of the input parameters is given along with several suggestions for possible alterations of the GRG main program and subroutines in order to revise data and output. An example problem is given. Beginning with the objective and constraint equations a step by step solution is given including the actual user supplied subroutines and data cards. Actual computer output is also reproduced.

2. Implementation

To implement GRG the user must supply one mandatory subroutine, GCOMP. Two additional routines, PARSH and SUMRY, may be provided if the user so desires. Arguments for all subroutines are double precision.

a. GCOMP (G,X)

GCOMP computes the constraint functions and the objective function at the x(•) vector which contains the N variables at their present values. The array G(•) then returns these values from GCOMP to the internal subroutines. No COMMON area is necessary although it is available if the user needs additional variable values for the computations. Any coefficients may be supplied by data cards.

b. PARSH (X,G,M,N,GRAD)

PARSH computes the gradients for each of the constraints and the objective function and returns these values to the main program in the (M + 1) x N array GRAD(•). Any coefficients needed for derivative computations can be held in COMMON with GCOMP. If analytical derivatives are not highly nonlinear or are extremely expensive to evaluate GRG contains a PARSH routine of its own which uses forward numerical differencing to establish the derivatives.

c. SUMRY (N,M,X,G)

This subroutine is available if the user desires any additional calculations and printing after GRG has arrived at a final solution. For any data in addition to the argument list COMMON blocks can be used. For example, the Lagrange multipliers are available for checking.

Using format statements which are preset by GRG the user must supply about ten cards containing parameters and tolerances for determining when final convergence has been obtained. In addition initial values for the x(•) vector must be supplied and any variables having finite upper and lower bounds must be listed on separate cards.

3. Printout

GRG initially prints out a summary of all initial parameters, tolerances and initial x(•) vector values along with their upper and lower bounds. Several levels of output are available. It is possible to receive, in very readable form, the objective function value, constraint function values, x(•) vector values, gradient vectors and other computational aids such as stepsize and explicitly viclated constraints. These values can be requested after each iteration or only after the final solution. The last printout delivers final values, binding constraints and number of calls for function and gradient evaluations.

4. Debugging Aids

In much the manner of SUMT, GRG provides a means of checking derivatives by two methods. These aids of course won't catch every programming error but they do help

in determining if the problem is in the function evaluation or the gradients. To check GRG gradients it is necessary to alter the output level and run the problem twice, first with the user's PARSH routine with analytical derivatives and again with the PARSH that is an internal subroutine and delivers derivatives by use of numerical differencing. GRG then prints out the initial gradient values for all equations and stops without attempting to solve the problem until the user is satisfied the gradient and function evaluation subroutines are consistent.

C. GRAVES

The GRAVES code was developed by G. W. Graves of U.C.L.A. and has been used over a period of twelve years to solve a wide variety of real world problems. The GRAVES nonlinear program is a linear approximation algorithm of the feasible direction class. GRAVES specifically solves the problem:

$$\text{minimize} \quad g^{M}(y_1, y_2, \ldots, y_{N3})$$

$$\text{subject to} \quad g^{i}(y_1, \ldots, y_{N3}) = 0, \quad i = 1, \ldots, M2$$

$$g^{i}(y_1, \ldots, y_{N3}) \leq 0, \quad i = M2+1, \ldots, M$$

with

$$LB_i \leq y_i \leq UB_i \qquad i=1, \ldots, N3$$

$g^{i}(y)$ $(i = 1, \ldots, M)$ must be continuously

differentiable functions.

The GRAVES code solves general nonlinear programs by solving a sequence of local linear problems which in the limit converge to a local stationary point. The local linear stationary points are global or local minima, or saddlepoints if the original nonlinear system is feasible. However, the test of practicality of an algorithm is not the existence of convergence but the speed with which a solution is obtained. This algorithm has been developed over the years and has evolved in response to actual problems. The code has performed well when used by its developer and analysts familiar with the extensive interior features, options and tuning mechanisms.

Linear programming is applied repeatedly to the linearized nonlinear problem in such a way that the solutions to the linear subproblems can converge to the sclution of the nonlinear problem. Linear approximations of nonlinear functions are obtained by using first order Taylor series approximations expanded at $y(\bullet)$ to replace the nonlinear functions in the original problem. The code has been termed a "local," "gradient," "stepwise" correction descent algorithm because of the method of selection of new points in the minimization process. Given a point y^0 the next "step" is $y = y^0 + k\Delta y$ where the step length is determined by the scalar k which is based on the behavior of the system in the neighborhood of the current point y^0. The step direction is determined by the gradients of the functions $g^i(y)$ $(i=1,\ldots,M-1)$. The direction of improvement is obtained by estimating the remainder term in the Taylor series approximation to $g(y^0 + \Delta y)$ and solving the

associated local linear programming problem. The remainder terms make the local linear approximations more sensitive to the behavior of the approximated functions because they are measures of the magnitude of local nonlinearity [Ref. 6].

The generic local linear programming problem is:

Subject tc the constraints

$$\nabla g^{i}(y^{o})\Delta y \leq -g^{i}(y^{o}) - kr^{i} \qquad i=1,\ldots,m-1$$

choose y to minimize $\nabla g^{m}(y^{o})\Delta y$.

The r^{i} are constants used in place of the Taylor series remainders (for convenience these are estimated from function behavior over the most recent step) and are equal to zero for strictly linear functions. The k is used to parametrically adjust the solutions of the local linear problems; as k decreases, improvement in feasibility becomes easier but the gain in the nonlinear problem decreases. This scaling parameter is also used to insure that once a feasible point has been located each successive point is also feasible.

After solving each local linear program and calculating the Taylor series remainders an interval determination is used on each constraint to determine a feasible movement interval. Each constraint is replaced by a quadratic Taylor approximation for determining the interval of feasibility along the Δy solution of the local linear program. If feasibility is impossible along Δy within the approximation range of the functions, the interval is modified to that corresponding to 1/2 the current solution infeasibility, then 3/4 the current infeasibility then 7/8 etc. until a

nonempty interval is found. The minimum of the objective function over this interval is then estimated by one of the optional ray search mechanisms, a new solution is obtained and another local linear program is begun.

The solution of a particular nonlinear problem requires the user to specify (or use the default values for) several of the algorithm's parameters including:

ZL — a "zero level" for variables (used throughout the nonlinear and linear programming routines)

ZT — a "zero tolerance" for functions

TL — a "tolerance level" for $G^{M}(y)$ used for termination of solutions by bounding the objective

IX1 — a "print priority" for intermediate output during solution

IX2 — a "problem type indicator" for problems with special structure (for instance, pure L.P.)

IPC, IPN — permutation limits for variable classes in hybrid pricing schemes (for large scale problems)

ICYCLE — a nonlinear iteration limit between calls to the user control modules

NEGF — pricing mechanism control (e.g. first negative, most negative, or candidate set

size for large scale problems)

JM — a ray search mode selector (e.g. quadratic fit, lattice/binary search, etc.)

RELAX — a relaxation mechanism to allow weighted gains in the objective function for infeasible intermediate solutions

KERNF — a kernel flush threshold for relaxation solutions (minimum explicit basis dimension in the local L.P. before gain check)

EB — the "equation bandwidth" to permit movement of solution in the presence of nonlinear equality constraints.

AR — the "approximation range" over which functions will be expected to behave reasonably with respect to second order Taylor series approximation

NP — the "number of points" for the direct ray search lattice option

IMIN — the "number of (binary) minimizations" for the direct ray search option

M, N3, M2 — the problem dimensions

Convergence for the algorithm may be rigorously proved [Ref. 7] for the usual class of quadratic problems (necessary assumptions are required for all problems concerning Kuhn-Tucker regularity and bounded steps from the local linear programming problems). Although the algorithm is a first order descent method, second order problem

representations can be accomodated [Ref. 8]. The imbedded linear programming algorithm has been described in Ref. 9; it is designed to facilitate primal and dual manipulation of the local linear programming problems, (especially basis changes brought about by parametric adjustment of the right hand side via k) as well as to deal with degeneracy, local inconsistencies and locally unbounded solutions.

The termination criterion for the code is a vector y' which satisfies

$$g^{M}(y') \leq g^{M}(y) + e$$

where e is a small user supplied value (TL). An e value which is scaled to the magnitude of the objective function $g^{M}(y')$ is very important to prevent unreasonable consumption of computer time when the gain in convergence is very small for each new solution. A maximum nonlinear iteration count (ICYCLE) is also used to interrupt progress.

1. Documentation

The GRAVES code has not been released to the general public and presently little documentation exists that is devoted explicitly to detailed discussion of the algorithm. Reference 10, "Sortie Allocation by a Nonlinear Programming Model for Determining a Munitions Mix", by R.J. Clasen, G.W. Graves, and J.Y. Lu is a report published by the Rand Corporation in 1974 for the Department of Defense to study the maximization of effectiveness of tactical sorties assigned to air-to-surface missions. The Rand study used a PL/1 version of the GRAVES method but a FORTRAN IV double precision version of slightly later vintage was utilized for this comparison. Chapters III and IV of the

Rand document are devoted to the theoretical background of this algorithm and the application of the programming code to the specific tactical air support problem. Appendices A and B present the alterations necessary to utilize the code for other nonlinear problems. A flowchart of the logic and the calculations performed by the code is presented along with a listing of the internal subroutines and variables. Variables which may or must be set by the user are denoted along with default values. Appendix C describes an additional program called CONTEST which is available for checking the consistency of the user's subroutines. A flowchart for the CONTEST program is included.

2. Implementation

The user of the GRAVES code must supply four FORTRAN subroutines: SETUP, FCNGEN, COLGEN and RESET.

a. SETUP (INAM)

SETUP is called only once at the start of each nonlinear solution attempt by the main GRAVES code. It does all of the data reading for the main code and provides the coefficients for function and derivative evaluation in other user supplied subroutines. SETUP shares COMMON blocks with the GRAVES internal subroutines. All variable input is done in the SETUP subroutine. All the values for all initial points and parameters must be input here; the user can call any outside FORTRAN program of his own for input of these values. Any data structure may be employed that will provide information for the following function generator routines.

b. FCNGEN (JP)

FCNGEN is called to calculate only $g^M(y)$ (the objective function) if JP = 1. If JP = 0, then all M functions will be evaluated and stored in the array G(•). G(1), G(2),...,G(M-1) are the values of the constraints and G(M) is the objective function value.

c. COLGEN (JC)

COLGEN computes the gradient of the variable JC for each constraint and the objective function. On call the procedure must place the JCth column of the gradient in CA(1), CA(2),...,CA(M). The method of determining the derivative may be analytical or a numerical differencing routine. The value of the primal variables y(•) will not have changed since the last FCNGEN call, thus the function values from that call are still valid.

d. RESET (LPC)

RESET is called only after the internal control routines have determined that a termination condition exists. This allows the user to diagnose the nature of the termination, to print any additional output that he needs, or closely monitor and interact with the solution progress. The user maintains complete control of the code and it is possible to restart the solution, alter the initial point, change functions or tuning parameters, or even run a continuing sequence of nonlinear problems.

3. Printout

The output level can be varied by use of a parameter in SETUP. The amount of output can be controlled from extremes of none to voluminous detail within the local linear programming solutions. If not inhibited the GRAVES

code prints the initial parameter values and tolerance levels. Final output typically includes the nonzero variables, the final values for the objective and constraint functions as well as the Lagrange multipliers if the user desires.

4. Debugging Aids

GRAVES provides an additional main program named CONTEST (consisting of only about 100 FORTRAN statements) which can be used to debug the user's SETUP, COLGEN and FCNGEN routines for mathematical consistency and logical program compatability. The program, acting as a surrogate nonlinear package, prints the derivatives for the variables and compares these to the results from differencing. Initial values for $y(\circ)$ and evaluations of objective and constraint functions are also provided for checking. CONTEST also checks for individual function linearity and convexity, properties of potential use to the analyst, and thus (after providing information for the debugging of the user subroutines and data) can also be utilized to gain an intuitive feel for the problem. CONTEST can provide the user with the appropriate zero and tolerance levels, approximate range for the Taylor series over the specified domains and an insight as to the degree of nonlinearity of each constraint .

III. METHOD AND RESULTS OF STUDY

The first step in the study was to obtain the three codes and documentation for each. The codes were permanently installed on a program library at the NPGS Computer Center and subsequently only the user subroutines and initial parameters were needed for each problem.

Of the eight problems five were received on computer cards from Professor Lasdon already encoded for GRG with the optimal parameter settings already determined. This degree of professional tuning was not really desirable from an experimental point of view but the GRG solutions for the three remaining user coded problems were similar in accuracy and time so this bias was actually found to be minimal. A sixth problem was selected from the Himmelblau collection [Ref. 11] and the remaining two were adaptations of an inventory model and an entropy model which were specifically designed to illustrate real world problems with few constraints but many independent variables. The test problems selected were considered typical of the small to medium size problems being solved today. The structure and degree of difficulty among the eight are quite varied and represent a fair sample of available relevent problems.

No attempt was made to keep an accurate accounting of the man hours necessary for preparation of the problems for each code since any analysis along these lines would be extremely biased because of the learning curve for each problem and the head start given in this respect to GRG. Once subroutines for evaluating the objective and constraint functions and the gradients had been coded for one of the nonlinear programs it was quite simple to adapt them to either of the other two codes. For most of the problems the GRAVES code was utilized for the initial programming because

of its superior turnaround time which is made possible by its very low memory requirements. Turnaround time may be of small importance for comparing the results of several different initial starting vectors or termination parameters but for the initial coding and debugging phase quick turnaround is highly desirable. In addition the GRAVES code has the special routine CONTEST, which is a main program rather than a subroutine, for checking the feasibility and consistency of the function and gradient subroutines without utilizing the main nonlinear package. Although SUMT and GRG have accurate and easy to use procedures for checking the consistency of the gradient and function evaluation subroutines they both require utilization of the main nonlinear codes with their high core and corresponding time requirements.

Himmelblau estimated preparation times for a typical problem and experienced user of from one to six hours for both GRG and SUMT [Ref. 11 pg. 381]. SUMT preparation times for unfamiliar users were considered to be two to five times as great. These figures are based on problems with less than 50 variables and are in keeping with this author's experience. Occasionally coding errors arose which required literally days to debug but these were oversights of the user and did not appear to be inherent to any one particular code.

The SUMT code required a third computational subroutine to evaluate the Hessian matrix and this subroutine required on the average a great deal more time than the gradient and function subroutines. For extremely nonlinear problems the calculation, coding and debugging of second derivatives was tedious and there were many opportunities for errors.

The GRG and GRAVES code require comparable preparation time for the computation subroutines although parameter

setting and alteration for the GRAVES code was considerably quicker and easier. All tolerances and initial values could be input in the GRAVES subroutine SETUP through the use of individual cards or Do Loops. The GRG code requires a complete listing in a pre-set format of all initial x(•) values as well as any upper and lower variable bounds that need to be provided. The format for these bounds allows only one bound per card thus entailing 200 separate cards for a hundred variable problem. Both SUMT and GRG use parameter and option cards which are difficult to initiate, alter and comprehend. With the exception of the two or three parameters which were consistently altered for each problem it was necessary to carefully consult the user's manual in order to change any of the tolerances for tuning a final solution. The GRG default values worked quite well for most of the test problems (which may be due to the tuning performed by the originators) but when the need for alteration arose the manual was a necessity.

The documentation for both the SUMT and GRG codes was excellent. Instructions for initial deck arrangements and the parameters required along with the recommended values for the tolerances were clear and sufficient. The GRAVES code is not presently a publicly distributed package and as such the documentation is not comparable to SUMT and GRG in regards to the proper tolerances and parameters to be utilized.

The SUMT and GRG codes are production type codes which are intended to solve a variety of problems without the user attempting to make any significant changes in the logic and methods of the codes. The GRAVES code is intended to be a production code of a personalized nature which requires that the analyst be aware of the internal logic and be able to interact with the code. Although the code has been steadily expanded over the years to solve a constantly enlarged list

of real world problems (in fact, other versions have been produced for nonlinear integer GUB problems) the alterations have been enacted to solve specific problems without sufficient attention given to the global effect on all problems in a given class.

Although not planned as part of the original study, Professor Graves was contacted and did provide recommended parameter settings for several of the problems based on a description of the performance of his package on early trials. These suggestions measurably increased the efficiency of the code.

After successful solutions were obtained for all the codes, parameters and tolerances were adjusted in the codes in order to obtain comparable objective function values and constraint tolerances. In most cases the GRG solutions were allowed to stand and the SUMT and GRAVES codes were adjusted. This procedure was adopted because of the prior tuning of the GRG problems. In most instances the GRAVES code did not achieve the same level of accuracy as GRG while SUMT tended in some cases to compute for several iterations without showing a significant increase in solution accuracy. By terminating SUMT at the accuracy level of GRG, CPU time in excess of 15% was saved in some problems.

Once comparable levels of precision had been attained two new initial vectors were selected for each problem and the algorithms were run again. The objective function values and solution times for all three initial points are listed in Table III-1.

Table III-1. Solutions and times from three initial starting points

Test problems run on an IBM 360/67 using FORTRAN H compile for main code and FORTRAN G compile for the user subroutines.

	GRG	SUMT	GRAVES
Problem 1			
f(x)	-1866.2	-1910.0	-1906.1
Time	36	103	4
f(x) a	-1882.1	-1910.4	-1906.1
Time	35	114	3
f(x) b	-1875.0	-1910.4	-1906.1
Time	26	99	4
Problem 2			
f(x)	-47.75	-47.76	-47.76
Time	2.57	9.34	14.83
f(x) a	-47.76	-47.73	-47.75
Time	3.22	10.50	13.20
f(x) b	-47.76	-47.73	
Time	2.49	9.93	
Problem 3			
f(x)	32.35	32.35	32.92
Time	6.63	13.70	37.07
f(x) a	32.35	32.35	32.88
Time	6.80	13.71	73.80
f(x) b	32.35	32.35	54.99
Time	7.04	13.99	51.72

	GRG	SUMT	GRAVES
Problem 4			
f(x)	-0.866	-0.866	-0.862
Time	3.07	6.7	14.40
f(x) a	-0.866	-0.675	-0.861
Time	2.11	6.70	12.12
f(x) b	-0.866	-0.866	-0.675
Time	4.24	6.31	14.71
Problem 5			
f(x)	0.0557	0.0557	0.0557
Time	13.85	385.00	8.85
f(x) a	0.0567	0.0557	
Time	11.05	393.00	
f(x) b	0.0557	0.0557	
Time	14.15	393.00	
Problem 6			
f(x)	-1735.0	-1735.0	-1720.0
Time	51	123	53
f(x) a	-1733.0	-1735.0	-1722.0
Time	48	111	49
f(x) b	-1731.0	-1735.0	-1723.0
Time	50	116	49

	GRG	SUMT	GRAVES
Problem 7			
f(x)	80.893	80.728	193.032
Time	96	246	234
f(x) a	91.191	80.728	376.144
Time	120	166	233
f(x) b	263.639	80.728	527.614
Time	49	161	233
Problem 8			
f(x)	-3.40	-3.47	-3.46
Time	32	54	150
f(x) a	-2.96	-3.47	-3.46
Time	21	54	142
f(x) b	-3.41	-3.47	-3.21
Time	44	55	59

a,b initial points for alternate solutions are listed with respective problem description in Appendix A.

Efficient computer coding of the user supplied subroutines will be a very important consideration in the amount of CPU time a particular algorithm requires and an attempt was made to determine which code is most dependent on the programming capabilities of the user. The GRG code records and prints as part of the final output the number of calls that were made for function and gradient evaluations. Similar counters were inserted in SUMT and GRAVES and the total calls for each are presented in Table III-2.

Obviously because of the inherently different approaches taken for gradient evaluations by the codes these numbers alone do not represent an accurate picture of the importance of the user supplied subroutines. While SUMT calls for the gradient of one constraint at a time and GRAVES evaluates the gradient of one variable at a time relative to all the constraints, GRG evaluates all constraints in one call necessitating a N x N matrix and thus showing only one gradient call for each N calls made by SUMT and GRAVES. In addition SUMT requires a Hessian routine while GRAVES and GRG have no comparable evaluation requirement.

Table III-2. Subroutine calls for each code

Problem		SUMT [a]	GRG [b]	GRAVES [c]
1	F	15929	257	26
	G	1479	54	2609
	H	1479		
2	F	2432	172	412
	G	156	13	8681
	H	156		
3	F	4982	546	964
	G	414	35	39407
	H	414		
4	F	9151	244	520
	G	656	18	1425
	H	656		
5	F	44255 #	303	28
	G	92127 #	31	1851
	H	3234		
6	F	3193	375	449
	G	310	69	52397
	H	310		
7	F	3447	1605	200
	G	147	76	301934
	H	84		
8	F	563	223	603
	G	1320 #	13	63144
	H	28		

F-function calls G-gradient calls H-hessian calls
#-numerical differencing was used for some constraints

Notes for Table III-2.

a-Each SUMT function call and gradient call required evaluating only one of the M constraints. Thus, M+1 subroutine calls are needed to obtain the same results as one GRG subrcutine call.

b-Each GRG function and gradient call required evaluating all M constraints and the objective function.

c-Each GRAVES function call requires either all constraints and the objective function or the objective function alone. Each gradient call evaluates the gradient column for all M constraints and the objective function associated with one of the N variables, thus N gradient calls are required to obtain the same results as one GRG gradient call.

In order to present an accurate evaluation of the importance of the user's efficiency in programming while avoiding any misguided attempt to standardize the number of function and gradient calls for the different codes each problem was re-run a second time using an added Do Loop in each user subroutine. This loop required the evaluation of each subroutine twice whenever it was called by the main nonlinear programming code. The increase in time for each problem represents a fair estimation of the total CPU time being spent in the user subroutine. These results are shown in Table III-3. Each problem's percentage is an average for the three different starting points that were used for that problem.

Table III-3.

Proportion of compute time spent in user subroutines

Problem	SUMT	GRG	GRAVES
1	.231	.042	.373
2	.800	.459	.507
3	.419	.465	.083
4	.093	.154	.212
5	.886 #	.332	.569
6	.111	.412	.723
7	.108	.362	.413
8	.704 #	.938	.635

#-numerical differencing was used for some constraints

A. Algorithms' performance for each problem:

The actual objective and constraint equations for each problem along with the results for each nonlinear programming code are included in Appendix A.

Problem 1

Problems 1 and 2 are both examples of determining the chemical composition of a complex mixture under conditions of chemical equilibrium. Problem 1 included 45 independent variables and 16 linear equality constraints. SUMT and GRAVES returned the best solutions and GRAVES solved the problem approximately 25 times faster than SUMT. SUMT and GRAVES obtained identical results for all three initial points while GRG fluctuated slightly. The

objective function and derivative subroutines for Problem 1 were demonstrated by Professor Graves to be inconsistent as x_{jk} approaches zero and he provided a special alteration of this problem for his code, which requires continuously differentiable functions over the entire bounded region of $y(\bullet)$.

Problem 2

Problem 2 was also a chemical equilibrium problem which had been redefined in the Himmelblau study from a Bracken and McCormick problem [Ref. 12]. All three codes solved the problem handily with GRG having the best time. GRAVES was not able to obtain a feasible point from one of the alternate initial points.

Problem 3

Problem 3 was formulated by the Shell Development Co. for the original Colville study and consisted of 15 variables and 5 nonlinear inequality constraints. SUMT and GRG returned identical solutions for all three initial vectors but the GRAVES solution was less precise.

Problem 4

The problem was maximize the area of a hexagon in which the maximum diameter was unity with 9 independent variables, 13 nonlinear inequality constraints and a lower bound of zero for x_9. The consistent results and low times in Table III-1 belie the difficulties encountered by the codes. GRG consistently returned with a local minimum value of about -0.4 (-0.866 is the global

minimum) until a phone call to Professor Lasdon resulted in an alteration to one of the tolerances utilized in the internal GRG subroutine DEGEN. Without this professional aid from one of the co-founders of the code it is doubtful that GRG would have ever attained the global minimum. GRAVES was not able to attain the same level of accuracy as SUMT and GRG. This problem contains many local minima and SUMT and GRAVES each converged to -0.675 from one of the alternate initial points.

Problem 5

Problem 5 is probably the most difficult test case in this study. It includes a linear objective function, 24 variables, 12 nonlinear equality constraints, 2 linear equality and 6 nonlinear inequality constraints. The independent variables are also bounded to positive values. GRG returned a quick and accurate solution from three different starting points. However, with an initial point $x_i = 0.4$ GRG was not able to obtain a feasible point while SUMT was able to reach the global minimum from all of these points. SUMT, which is not expressly designed to handle nonlinear equality constraints, required over six minutes for each of these solutions. GRAVES, which appears to be more sensitive to the initial point than the other two codes had the best solution time from the original point but was not able to locate a feasible solution when using the alternate starting points.

Because of the complexity and nonlinearity of the first 12 constraints SUMT's numerical differencing subroutine DIFF2 was utilized to compute the Hessian matrix for these constraints.

Problem 6

Problem 6 was a weapon assignment problem with 100 independent variables, a nonlinear objective function, 12 linear constraints and zero lower bounds for the variables. All three codes returned approximately equal objective solutions although the actual variable values were quite diverse. GRAVES and GRG were twice as fast as SUMT.

Problem 7

This problem was adapted from an inventory model created by D.A. Schrady and U.C. Choe [Ref. 13]. The $x_i(\bullet)$ $(i=1,\ldots,50)$ represent the reorder quantity for 50 inventory items and $x_i(\bullet)$ $(i=51,\ldots,100)$ represent the reorder points for the same 50 items. SUMT as usual obtained consistent results from all three initial starting points but GRG encountered severe numerical problems from the alternate starting points $x_i(\bullet) = 10$ and $x_i(\bullet) = 1000$ $(i=1,\ldots,100)$ and as a result produced correspondingly inferior sclutions. GRAVES was not able to reach the global minimum apparently because of the utilization of an external FORTRAN subroutine to approximate the culmulative normal distribution. This subroutine was accurate only to 10^{-7} and the gradient calculations showed inconsistencies because of this lack of precision. Again to guarantee convergence the GRAVES algorithm requires continuously differentiable functions.

Problem 8

Problem 8 was adapted from an entropy model proposed by A.J. Scott [Ref. 14]. The nodes in Figure A-1

illustrate 46 population centers connected by a transportation network, represented by the connecting arcs. Using a congestion cost function the model yields an equilibrium solution that identifies nodal populations as entropic functions of the total cost of the journey to work. All three codes returned similar solutions with SUMT leading the way while GRG made its worst showing. Starting with $x_i(\bullet) = 10.87$ $(i=1,\ldots,46)$ GRG located a local minimum of -2.96 (global minimum was -3.47). SUMT was generally twice as fast as GRAVES when they both returned global minima.

B. Conclusions and Summary:

Perhaps the easiest way to summarize the results of this analysis is to return to the evaluation criteria which were listed in Chapter 1.

1. Accuracy of the final solutions:

SUMT was able to attain the best solution for all eight problems while GRG fell short on Problem 1 and Problem 8. GRAVES found the global minimum in five of the eight problems.

2. Robustness:

Again SUMT was superior going to a local minimum only once (Problem 4) while GRC had variances in the precision attained for Problems 1,5,6,7,8. GRAVES in two problems was not able to locate a feasible point when starting from one of the alternate initial vectors.

3. Speed of Convergence:

SUMT, while very reliable, was also very

deliberate and could not compare with GRG in the CPU time category. GRG was anywhere from 3 to 30 times as fast as SUMT. The solution times for GRAVES were inconsistent but for problems 1 and 5 GRAVES was from 1 to 9 times faster than GRG.

4. Ease of preparation for user subroutines:

SUMT required the most effort because of the need to evaluate the Hessian matrices. GRAVES was considered the easiest to use because of its superior technique employed for initiating and altering tolerances.

5. Aids for Debugging:

All three codes provided fine methods for comparing the consistency of the gradient and function subroutines but GRAVES was clearly superior because of its additional main program CONTEST which required only about 95K bytes of main memory for 100 variables and 50 constraints.

6. Readability of final and intermediate output:

All three codes have very comprehensive output but SUMT only allows two levels of printout while GRG and GRAVES have six possible levels.

7. Failure mode:

The GRG code would explain in very concise but clear language the reason for terminating its solution attempts. A simple cure for most SUMT difficulties was to give it more time. GRAVES had many options available for the user to employ when the solution was not forthcoming. Unfortunately there were too many options and the user at times was overwhelmed with possible alternatives.

All three codes have advantages and disadvantages with the choice of which to use being controlled by each problem and user's special circumstances. SUMT is always very accurate. GRG is usually fast, quite accurate but requires a substantial amount of memory space. The GRAVES code when properly tuned provided tremendous results with low time and memory requirements.

The GRG code, except in computer centers with restricted memory availability, is probably the superior choice of these three codes although SUMT was able to consistently attain the global minima from all initial points. Although not a generally distributed code, because of its low core requirements, the GRAVES code seems to hold promise for large problems.

APPENDIX A

TEST PROBLEMS AND RESULTS

Problem 1

Source: A.P. Jones, "The Chemical Equilibrium Problem: An Application of SUMT," Research Analysis Corporation, McLean, Va., RAC-TP-272, 1967 [Ref. 15].

No. of variables: 45

No. of constraints: 16 linear equality constraints

Objective function:

$$\text{Minimize: } f(x) = \sum_{k=1}^{7} \left[\sum_{j=1}^{n_k} x_{jk} \left(c_{jk} + \ln \frac{x_{jk}}{\sum_{j=1}^{n_k} x_{jk}} \right) \right]$$

Constraints:

$$h_i(x) = \sum_{k=1}^{7} \left(\sum_{j=1}^{n_k} E_{ijk} x_{ijk} \right) - b_i = 0 \qquad i = 1,\ldots,16$$

$$x_{jk} \geq 0 \qquad j = 1,\ldots,n_k \qquad k = 1,\ldots,7$$

b_j's and c_{jk}'s for Problem 1

i	b_i	j	k	c_{jk}	j	k	c_{jk}
1	0.6529581	1	1	0.0	6	3	0.0
2	0.281941	2	1	-7.69	7	3	2.2435
3	3.705233	3	1	-11.52	8	3	0.0
4	47.00022	4	1	-36.60	9	3	-39.39
5	47.02972	1	2	-10.94	10	3	-21.49
6	0.08005	2	2	0.0	11	3	-32.84
7	0.08813	3	2	0.0	12	3	6.12
8	0.04829	4	2	0.0	13	3	0.0
9	0.0155	5	2	0.0	14	3	0.0
10	0.0211275	6	2	0.0	15	3	-1.9028
11	0.0022725	7	2	0.0	16	3	-2.8889
12	0.0	8	2	2.5966	17	3	-3.3622
13	0.0	9	2	-39.39	18	3	-7.4854
14	0.0	10	2	-21.35	1	4	-15.639
15	0.0	11	2	-32.84	2	4	0.0
16	0.0	12	2	6.26	3	4	21.81
		13	2	0.0	1	5	-16.79
		1	3	10.45	2	5	0.0
		2	3	0.0	3	5	18.9779
		3	3	-0.50	1	6	0.0
		4	3	0.0	2	6	11.959
		5	3	0.0	1	7	0.0
					2	7	12.899

E_{ijk} Data for Problem 1

x_{jk}^{i}	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
x_{11}	1															
x_{21}		1														
x_{31}			1													
x_{41}				1	1											
x_{12}	1															
x_{22}		1														
x_{32}			1													
x_{42}				1								1				
x_{52}					1							-1				
x_{62}						1						-1				
x_{72}							1					1				
x_{82}								1				1				
x_{92}				1	1											
$x_{10,2}$		1			1							-1				
$x_{11,2}$		1		1	1											
$x_{12,2}$		1		-1	1							-2				
$x_{13,2}$									1			-1				
x_{13}	1															
x_{23}		1														
x_{33}			1													
x_{43}				1												
x_{53}					1											
x_{63}						1										
x_{73}							1									
x_{83}								1								
x_{93}				1	1											
$x_{10,3}$		1			1											
$x_{11,3}$		1		1	1											
$x_{12,3}$		1		-1	1											
$x_{13,3}$										1						

x_{jk}^{i}	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$x_{14,3}$											1	-4				
$x_{15,3}$	1										1	-3	-1			
$x_{16,3}$	2										1	-2	-2			
$x_{17,3}$	3										1	-1	-3			
$x_{18,3}$	4										1		-4			
x_{14}				1									1			
x_{24}													1			
x_{34}				-1									1		-4	
x_{15}				1										1		
x_{25}														1		
x_{35}				-1										1		-4
x_{16}															1	
x_{26}		1		-1											1	
x_{17}																1
x_{27}		1		-1												1

Alternate Initial Points:

a) $x_{jk} = 0.01 \qquad j=1,\ldots,n_k \qquad k=1,\ldots,7$

b) $x_{jk} = 1.0 \qquad j=1,\ldots,n_k \qquad k=1,\ldots,7$

Results for Problem 1

	initial	GRG	SUMT	GRAVES
f(x)	-30.958	-1866.2	-1910.4	-1906.1
x_{11}	0.1	1.3E-01	1.8E-06	4.0E-02
x_{21}	0.1	2.7E-01	2.5E-01	2.4E-01
x_{31}	0.1	3.3E 00	3.7E 00	3.5E 00
x_{41}	0.1	1.7E 01	2.5E-01	2.0E-05
x_{12}	0.1	5.2E-01	6.5E-01	6.1E-01
x_{22}	0.1	0.0	1.2E-03	2.0E-05
x_{32}	0.1	4.2E-01	4.0E-04	2.0E-05
x_{42}	0.1	0.0	3.8E-07	2.0E-05
x_{52}	0.1	0.0	1.2E-06	2.0E-05
x_{62}	0.1	2.0E-02	7.2E-02	6.5E-02
x_{72}	0.1	5.0E-02	8.8E-02	8.3E-02
x_{82}	0.1	2.4E-02	3.5E-02	9.9E-04
x_{92}	0.1	1.8E 01	4.4E 01	2.9E 00
$x_{10,2}$	0.1	0.0	2.6E-02	2.0E-05
$x_{11,2}$	0.1	0.0	2.5E-02	2.0E-05
$x_{12,2}$	0.1	1.5E-02	4.1E-05	2.0E-05
$x_{13,2}$	0.1	1.6E-02	1.6E-02	1.6E-02
x_{13}	0.1	0.0	1.1E-07	2.0E-05
x_{23}	0.1	0.0	6.6E-05	1.7E-02
x_{33}	0.1	2.5E-02	3.6E-05	2.3E-01
x_{43}	0.1	0.0	1.8E-07	2.0E-05
x_{53}	0.1	0.0	4.5E-07	2.0E-05
x_{63}	0.1	6.0E-02	7.6E-03	1.5E-02
x_{73}	0.1	3.8E-02	2.5E-04	5.5E-03
x_{83}	0.1	2.4E-02	1.3E-02	4.7E-02
x_{93}	0.1	1.2E 01	2.4E 00	4.7E-02
$x_{10,3}$	0.1	3.2E-04	3.2E-03	4.4E 01
$x_{11,3}$	0.1	0.0	5.4E-07	2.0E-05
$x_{12,3}$	0.1	0.0	1.0E-05	6.6E-04
$x_{13,3}$	0.1	2.1E-02	2.1E-02	2.1E-02
$x_{14,3}$	0.1	2.2E-03	2.3E-03	2.0E-05

$x_{15,3}$	0.1	0.0	7.4E-07	1.0E-03
$x_{16,3}$	0.1	0.0	1.0E-07	2.0E-05
$x_{17,3}$	0.1	0.0	4.9E-08	2.0E-05
$x_{18,3}$	0.1	0.0	3.6E-08	1.3E-03
x_{14}	0.1	0.0	1.5E-07	2.0E-05
x_{24}	0.1	0.0	5.7E-07	2.0E-05
x_{34}	0.1	0.0	2.1E-06	6.1E-03
x_{15}	0.1	0.0	1.3E-07	2.0E-05
x_{25}	0.1	0.0	3.1E-07	2.0E-05
x_{35}	0.1	0.0	2.0E-06	2.0E-05
x_{16}	0.1	0.0	5.4E-06	2.3E-03
x_{26}	0.1	0.0	3.2E-06	2.2E-02
x_{17}	0.1	0.0	6.3E-06	2.0E-05
x_{27}	0.1	0.0	1.8E-06	2.0E-05
$h_1(x)$	0.647	-2.2E-15	7.2E-06	0.0
$h_2(x)$	0.818	6.6E-17	5.0E-06	0.0
$h_3(x)$	-3.405	1.9E-15	5.5E-06	0.0
$h_4(x)$	-46.70	7.5E-14	1.0E-05	0.0
$h_5(x)$	-45.93	6.9E-14	8.4E-06	0.0
$h_6(x)$	0.12	1.8E-16	2.7E-06	0.0
$h_7(x)$	0.112	2.8E-16	3.3E-06	0.0
$h_8(x)$	0.152	1.1E-17	2.5E-06	0.0
$h_9(x)$	0.085	1.7E-18	3.5E-06	0.0
$h_{10}(x)$	0.079	0.0	2.3E-06	0.0
$h_{11}(x)$	0.498	1.1E-17	2.0E-06	0.0
$h_{12}(x)$	-1.3	2.4E-15	-3.2E-07	0.0
$h_{13}(x)$	-0.7	0.0	1.6E-06	0.0
$h_{14}(x)$	0.3	0.0	2.4E-06	0.0
$h_{15}(x)$	-0.2	0.0	3.0E-07	0.0
$h_{16}(x)$	-0.2	0.0	1.9E-07	0.0
Time		36 sec	116 sec	5 sec

Problem 2

Source: D.M. Himmelblau, "Applied Nonlinear Programming," McGraw-Hill, Inc., New York, 1972, p. 396.

No. of independent variables: 10

No. of constraints: 3 nonlinear equality constraints

Objective function:

$$\text{Minimize: } f(x) = \sum_{i=1}^{10} \left[e^{x_i} \left(c_i + x_i - \ln \sum_{i=1}^{10} e^{x_i} \right) \right]$$

Constraints:

$$h_1(x) = e^{x_1} + 2e^{x_2} + 2e^{x_3} + e^{x_6} + e^{x_{10}} - 2 = 0$$

$$h_2(x) = e^{x_4} + 2e^{x_5} + e^{x_6} + e^{x_7} - 1 = 0$$

$$h_3(x) = e^{x_3} + e^{x_7} + e^{x_8} + 2e^{x_9} + e^{x_{10}} - 1 = 0$$

where $c_1 = -6.089$ $c_2 = -17.164$ $c_3 = -34.054$ $c_4 = -5.914$

$c_5 = -24.721$ $c_6 = -14.986$ $c_7 = -24.1000$

$c_8 = -10.708$ $c_9 = -26.662$ $c_{10} = -22.179$

Results for Problem 2

	initial	GRG	SUMT	GRAVES
$f(x)$	-21.015	-47.75	-47.76	-47.76
x_1	-2.3	-3.61	-3.17	-3.84
x_2	-2.3	-1.92	-1.88	-1.58
x_3	-2.3	-0.246	-0.254	-0.316
x_4	-2.3	-5.58	-6.58	-5.92
x_5	-2.3	-0.723	-0.723	-0.726
x_6	-2.3	-5.64	-7.19	-6.04
x_7	-2.3	-3.82	-3.61	-3.55
x_8	-2.3	-4.14	-4.02	-4.22
x_9	-2.3	-3.39	-3.22	-2.93
x_{10}	-2.3	-2.19	-2.31	-1.88
$h_1(x)$	-1.298	2.4E-05	4.6E-09	0.0
$h_2(x)$	-0.499	1.8E-05	-5.4E-09	0.0
$h_3(x)$	-0.398	8.0E-06	2.0E-09	0.0
Time		2.57 sec.	9.34 sec.	14.83 sec

Alternate Initial Points:

a) $x_i = 2.0 \qquad i=1,\ldots,10$

b) $x_i = -5.0 \qquad i=1,\ldots,10$

Problem 3

Source: Shell Development Co. (cited in Colville, IBM N.Y. Sci. Center Rept.320-2949, June, 1968, p. 22).

No of variables: 15

No. of constraints: 5 nonlinear inequality constraints

15 bounds on independent variables

Objective function:

$$\text{Maximize: } f(x) = \sum_{i=1}^{10} b_i x_i - \sum_{j=1}^{5} \sum_{i=1}^{5} yz - 2 \sum_{j=1}^{5} d_j z^3$$

where $y = c_{ij} x_{(10+i)}$

and $z = x_{(10+j)}$

Constraints:

$$2 \sum_{i=1}^{5} y + 3 d_j z^2 + e_j - \sum_{i=1}^{10} a_{ij} x_i \geq 0 \qquad j = 1,\ldots,5$$

$$x_i \geq 0 \qquad i = 1,\ldots,15$$

Data for Problem 3

j	1	2	3	4	5
e_j	-15	-27	-36	-18	-12
c_{1j}	30	-20	-10	32	-10
c_{2j}	-20	39	-6	-31	32
c_{3j}	-10	-6	10	-6	-10
c_{4j}	32	-31	-6	39	-20
c_{5j}	-10	32	-10	-20	30
d_j	4	8	10	6	2
a_{1j}	-16	2	0	1	0
a_{2j}	0	-2	0	0.4	2
a_{3j}	-3.5	0	2	0	0
a_{4j}	0	-2	0	-4	-1
a_{5j}	0	-9	-2	1	-2.8
a_{6j}	2	0	-4	0	0
a_{7j}	-1	-1	-1	-1	-1
a_{8j}	-1	-2	-3	-2	-1
a_{9j}	1	2	3	4	5
a_{10j}	1	1	1	1	1

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
-40	-2	-.25	-4	-4	-1	-40	-60	5	1

Alternate Initial Points:

a) $x_i = 5.0$ $\quad i=1,\ldots,15$

b) $x_i = 15.0$ $\quad i=1,\ldots,15$

Results for Problem 3

	initial	GRG	SUMT	GRAVES
f(x)	2400.1	32.349	32.349	32.92
x_1	1.0E-04	0.0	1.7E-06	0.0
x_2	1.0E-04	0.0	3.1E-05	0.0
x_3	1.0E-04	5.2	5.2	5.4
x_4	1.0E-04	0.0	4.4E-05	0.0
x_5	1.0E-04	3.06	3.06	3.07
x_6	1.0E-04	11.85	11.84	12.26
x_7	6.0E 01	0.0	1.6E-06	6.3E-04
x_8	1.0E-04	0.0	1.1E-06	0.0
x_9	1.0E-04	0.1	0.1	0.16
x_{10}	1.0E-04	0.0	8.9E-05	0.0
x_{11}	1.0E-04	0.3	0.3	0.29
x_{12}	1.0E-04	0.33	0.33	0.34
x_{13}	1.0E-04	0.39	0.40	0.40
x_{14}	1.0E-04	0.43	0.43	0.44
x_{15}	1.0E-04	0.22	0.22	0.24
$h_1(x)$	4.5E 01	4.9E-08	2.0E-04	1.4E-01
$h_2(x)$	3.3E 01	2.5E-11	1.8E-04	1.8E-01
$h_3(x)$	2.4E 01	4.9E-06	1.5E-04	8.8E-01
$h_4(x)$	4.2E 01	8.8E-08	1.4E-04	1.8E-01
$h_5(x)$	4.8E 01	1.8E-07	2.7E-04	4.5E-02
Time		6.93 sec	13.7 sec	37.07 sec

Problem 4

Source: J. D. Pearson, On Variable Metric Methods of Minimization, Research Analysis Corp. Rept. RAC-TP-302, McLean, Va., May, 1968 [Ref. 16].

No. of variables: 9

No. of constraints: 13 nonlinear inequality constraints

1 upper bound

Objective function:

Maximize:

$$f(x) = 0.5(x_1x_4 - x_2x_3 + x_3x_9 - x_5x_9 + x_5x_8 - x_6x_7)$$

Constraints:

$$1 - x_3^2 - x_4^2 \geq 0$$

$$1 - x_9^2 \geq 0$$

$$1 - x_5^2 - x_6^2 \geq 0$$

$$1 - x_1^2 - (x_2 - x_9)^2 \geq 0$$

$$1 - (x_1 - x_5)^2 - (x_2 - x_6)^2 \geq 0$$

$$1 - (x_1 - x_7)^2 - (x_2 - x_8)^2 \geq 0$$

$$1 - (x_3 - x_5)^2 - (x_4 - x_6)^2 \geq 0$$

$$1 - (x_3 - x_7)^2 - (x_4 - x_8)^2 \geq 0$$

$$1 - x_7^2 - (x_8 - x_9)^2 \geq 0$$

$$x_1x_4 - x_2x_3 \geq 0$$

$$x_3x_9 \geq 0$$

$$-x_5x_9 \geq 0$$

$$x_5x_8 - x_6x_7 \geq 0$$

$$x_9 \geq 0$$

Results for Problem 4

	initial	GRG	SUMT	GRAVES
f(x)	0.0	-0.866	-0.866	-0.862
x_1	1.0	-3.0E-05	-0.530	-0.014
x_2	1.0	2.0E-05	-0.333	-0.572
x_3	1.0	0.866	0.469	0.852
x_4	1.0	-0.5	-0.883	-0.512
x_5	1.0	0.0	-0.530	-0.015
x_6	1.0	-1.0	-0.848	-1.000
x_7	1.0	0.866	0.469	0.858
x_8	1.0	0.5	-0.369	-0.081
$h_1(x)$	-1.0	2.0E-13	4.9E-06	1.1E-02
$h_2(x)$	0.0	-6.6E-05	7.4E-01	8.2E-01
$h_3(x)$	-1.0	0.0	2.8E-07	7.9E-05
$h_4(x)$	0.0	-2.5E-05	5.8E-06	1.0E-03
$h_5(x)$	1.0	4.1E-05	7.4E-01	8.2E-01
$h_6(x)$	1.0	-2.5E-05	1.1E-05	0.0
$h_7(x)$	1.0	2.1E-13	1.2E-05	9.4E-03
$h_8(x)$	1.0	-1.5E-06	7.4E-01	8.1E-01
$h_9(x)$	0.0	-2.5E-05	1.1E-05	5.7E-03
$h_{10}(x)$	0.0	-2.7E-06	6.2E-01	4.9E-01
$h_{11}(x)$	1.0	0.866	2.4E-01	3.6E-01
$h_{12}(x)$	-1.0	-1.6E-19	2.7E-01	6.6E-03
$h_{13}(x)$	0.0	0.866	5.9E-01	8.6E-01
$h_{14}(x)$	1.0	1.0	5.1E-01	8.6E-01
Time		3.07 sec	6.7 sec	14.4 sec

Alternate Initial Points:

a) $x_i = -1.0$ $\quad i=1,\ldots,8$ $\quad x_9 = 0.0$

b) $x_i = 5.0$ $\quad i=1,\ldots,9$

Problem 5

Source: D.A. Paviani, Ph.D. dissertation, The University of Texas, Austin, Tex., 1969 [Ref. 17].

No. of variables: 24

No. of constraints: 12 nonlinear equality constraints

2 linear equality constraints

6 nonlinear inequality constraints

24 bounds on independent variables

Objective function:

$$\text{Minimize: } f(x) = \sum_{i=1}^{24} a_i x_i$$

Constraints:

$$h_i(x) = \frac{x_{(i+12)}}{b_{(i+12)} \sum_{j=13}^{24} x_j / b_j} - \frac{c_i x_i}{40 b_i \sum_{j=1}^{12} x_j / b_j} = 0 \qquad i = 1,\ldots,12$$

$$h_{13}(x) = \sum_{i=1}^{24} x_i - 1 = 0$$

$$h_{14}(x) = \sum_{i=1}^{12} \frac{x_i}{d_i} + f \sum_{i=13}^{24} \frac{x_i}{b_i} - 1.671 = 0$$

where $f = (0.7302)(530)\left|\frac{14.7}{40}\right|$

$$h_{(i+14)}(x) = \frac{-\left[x_i + x_{(i+12)}\right]}{\sum_{j=1}^{24} x_j} + e_i \geq 0 \qquad i = 1,2,3$$

$$h_{(i+14)}(x) = \frac{-\left[x_{(i+3)} + x_{(i+15)}\right]}{\sum_{j=1}^{24} x_j} + e_i \geq 0 \qquad i = 4,5,6$$

$$x_i \geq 0 \qquad i = 1,\ldots,24$$

Data for Problem 5

i	a_i	b_i	c_i	d_i	e_i
1	0.0693	44.094	123.7	31.244	0.1
2	0.0577	58.12	31.7	36.12	0.3
3	0.05	58.12	45.7	34.784	0.4
4	0.20	137.4	14.7	92.7	0.3
5	0.26	120.9	84.7	82.7	0.6
6	0.55	170.9	27.7	91.6	0.3
7	0.06	62.501	49.7	56.708	
8	0.10	84.94	7.1	82.7	
9	0.12	133.425	2.1	80.8	
10	0.18	82.507	17.7	64.517	
11	0.10	46.07	0.85	49.4	
12	0.09	60.097	0.64	49.1	
13	0.0693	44.094			
14	0.0577	58.12			
15	0.05	58.12			
16	0.20	137.4			
17	0.26	120.9			
18	0.55	170.9			
19	0.06	62.501			
20	0.10	84.94			
21	0.12	133.425			
22	0.18	82.507			
23	0.10	46.07			
24	0.09	60.097			

Results for Problem 5

	initial	GRG	SUMT	GRAVES
f(x)	0.14696	5.566E-02	5.566E-02	5.566E-02
x_1	0.04	2.8E-09	2.1E-07	0.0
x_2	0.04	1.1E-01	1.1E-01	1.1E-01
x_3	0.04	1.1E-01	1.1E-01	1.1E-01
x_4	0.04	0.0	1.3E-07	0.0
x_5	0.04	4.9E-14	2.8E-08	0.0
x_6	0.04	0.0	2.7E-08	0.0
x_7	0.04	7.6E-02	7.6E-02	7.6E-02
x_8	0.04	0.0	9.1E-07	0.0
x_9	0.04	0.0	6.1E-07	0.0
x_{10}	0.04	0.0	1.4E-07	0.0
x_{11}	0.04	0.0	1.3E-05	4.0E-07
x_{12}	0.04	1.1E-02	1.1E-02	1.1E-02
x_{13}	0.04	0.0	1.5E-06	0.0
x_{14}	0.04	1.9E-01	1.9E-01	1.9E-01
x_{15}	0.04	2.9E-01	2.9E-01	2.9E-01
x_{16}	0.04	0.0	1.1E-07	0.0
x_{17}	0.04	0.0	1.3E-07	0.0
x_{18}	0.04	0.0	4.0E-08	0.0
x_{19}	0.04	2.1E-01	2.1E-01	2.1E-01
x_{20}	0.04	0.0	3.7E-07	0.0
x_{21}	0.04	0.0	7.6E-08	0.0
x_{22}	0.04	0.0	1.5E-07	0.0
x_{23}	0.04	0.0	6.5E-07	0.0
x_{24}	0.04	4.1E-04	4.1E-04	4.1E-04
$h_1(x)$	-2.9E-01	-3.8E-08	1.3E-09	0.0
$h_2(x)$	2.2E-02	4.9E-07	-1.7E-09	0.0

	initial	GRG	SUMT	GRAVES
$h_3(x)$	-1.5E-02	1.7E-08	-1.4E-09	0.0
$h_4(x)$	2.8E-02	7.0E-20	7.2E-11	0.0
$h_5(x)$	-5.6E-02	1.7E-13	-9.0E-10	0.0
$h_6(x)$	1.1E-02	7.7E-21	-1.3E-09	0.0
$h_7(x)$	-2.4E-02	5.1E-07	9.6E-10	0.0
$h_8(x)$	5.9E-02	5.7E-27	1.5E-10	0.0
$h_9(x)$	4.3E-02	3.6E-35	1.9E-09	0.0
$h_{10}(x)$	4.1E-02	7.3E-21	-5.3E-11	0.0
$h_{11}(x)$	1.3E-01	-2.9E-33	3.6E-10	0.0
$h_{12}(x)$	1.0E-02	3.3E-09	6.0E-10	0.0
$h_{13}(x)$	-4.0E-02	-9.6E-17	-4.6E-10	0.0
$h_{14}(x)$	-7.3E-01	2.4E-11	4.8E-12	0.0
$h_{15}(x)$	1.6E-02	1.0E-01	1.0E-01	1.0E-01
$h_{16}(x)$	2.2E-01	-6.1E-10	2.6E-06	0.0
$h_{17}(x)$	3.2E-01	-9.8E-11	1.6E-06	0.0
$h_{18}(x)$	2.2E-01	1.2E-02	1.2E-02	1.2E-02
$h_{19}(x)$	5.2E-01	6.0E-01	6.0E-01	6.0E-01
$h_{20}(x)$	2.2E-01	3.0E-01	3.0E-01	3.0E-01
Time		13.85 sec	385 sec	8.85 sec

Alternate Initial Points:

a) $x_i = 0.08$ $\quad i=1,\ldots,24$

b) $x_i = 0.02$ $\quad i=1,\ldots,24$

Problem 6

Source: J. Bracken and G. P. McCormick, "Selected Applicatiens of Nonlinear Programming," John Wiley & Sons, Inc., New York, 1968, p. 26.

No. of independent variables: 100

No. of constraints: 12 linear constraints

100 lower bounds on the variables

Objective function:

$$\text{Minimize: } f(x) = \sum_{j=1}^{20} u_j \left(\prod_{i=1}^{5} a_{ij}^{x_{ij}} - 1 \right)$$

Constraints:

$$\sum_{i=1}^{5} x_{ij} - b_j \geq 0 \qquad j = 1,6,10,14,15,16,20$$

$$- \sum_{j=1}^{20} x_{ij} + c_i \geq 0 \qquad i = 1...,5$$

$$x_{ij} \geq 0 \qquad i=1,...,5 \qquad j=1,...,20$$

Data for Problem 6

	a_{ij}'s					b_j's	u_j's
i \ j	1	2	3	4	5		
1	1	.84	.96	1	.92	30	60
2	.95	.83	.95	1	.94		50
3	1	.85	.96	1	.92		50
4	1	.84	.96	1	.95		75
5	1	.85	.96	1	.95		40
6	.85	.81	.90	1	.98	100	60
7	.90	.81	.92	1	.98		35
8	.85	.82	.91	1	1		30
9	.80	.80	.92	1	1		25
10	1	.86	.95	.96	.90	40	150
11	1	1	.99	.91	.95		30
12	1	.98	.98	.92	.96		45
13	1	1	.99	.91	.91		125
14	1	.88	.98	.92	.98	50	200
15	1	.87	.97	.98	.99	70	200
16	1	.88	.98	.93	.99	35	130
17	1	.85	.95	1	1		100
18	.95	.84	.92	1	1		100
19	1	.85	.93	1	1		100
20	1	.85	.92	1	1	10	150
c_i	200	100	300	150	250		

Results for Problem 6

	initial	GRG	SUMT	GRAVES
x(*)	100.0	below	below	below
f(x)	-1755.0	-1735.0	-1735.0	-1720.0
$h_1(x)$	-1800.0	18.0	20.8	21.6
$h_2(x)$	-1900.0	-4.8E-13	6.4E-02	0.0
$h_3(x)$	-1700.0	12.0	11.2	37.7
$h_4(x)$	-1850.0	-3.4E-13	8.7	15.7
$h_5(x)$	-1750.0	6.8E-13	0.1	12.8
$h_6(x)$	470.0	18.0	6.8	17.5
$h_7(x)$	400.0	52.4	52.3	50.1
$h_8(x)$	460.0	9.1E-13	6.4E-02	0.0
$h_9(x)$	450.0	6.5E-13	1.7E-02	0.0
$h_{10}(x)$	430.0	1.1E-12	5.6E-02	0.0
$h_{11}(x)$	465.0	8.2E-13	3.1E-02	0.0
$h_{12}(x)$	490.0	8.3E-13	5.4E-02	0.0
Time		51 sec.	123 sec.	49 sec
Core		270K	194K	138K

Alternate Initial Points:

a) $x_{1j} = 10.0$ $x_{2j} = 5.0$ $x_{3j} = 15.0$
$x_{4j} = 7.5$ $x_{5j} = 12.5$ $j=1,\ldots,20$

b) $x_{ij} = 10$ $i=1,\ldots,5$ $j=1,\ldots,20$

Target j	Weapon Type i					
	1	2	3	4	5	Total
1	-8-		-5-		-40- (51) 48	-53- (51) 48
2	-6- (14) 14	-6- (2)	-9- (1) 37		-19- (43) 18	-40- (60) 69
3		-6-			-28- (48) 49	-34- (48) 49
4		-9- (23) 22	-12-		-34- (1) 2	-54- (24) 24
5		-11- (20) 18	-23-		-3- (1) 9	-37- (21) 27
6	-74- (99) 98		-10- (1) 2	-1-	-15-	-100- (100) 100
7	-33- (39) 37					-33- (39) 37
8	-24- (27) 27					-24- (27) 27
9	-20- (20) 21					-20- (20) 21
10	-15-		-10-		-40- (51) 52	-65- (51) 52
11				-13- (33) 27	-27- (1) 11	-40- (34) 38
12			-13-	-24- (41) 37	-9- 7	-46- (41) 44

Target j	Weapon Type i 1	2	3	4	5	Total
13				-13-	-33- (54) 54	-46- (54) 54
14	-5-	-3- 9	-11-	-46- (58) 41		-66- (58) 50
15	-10-	-20- (26) 26	-44- (44) 44	-16-	-1-	-91- (70) 70
16	-1-	-8- (24) 8	-8-	-36- (18) 45		-53- (42) 53
17		-10- (4) 16	-38- (71) 34			-48- (75) 50
18	3	-7-	-41- (57) 57			-48- (57) 60
19		-8-	-38- (64) 64			-46- (64) 64
20	-4-	-12-	-38- (62) 62		-1-	-55- (62) 62
Totals	-200- (199) 200	-100- (99) 99	-300- (300) 300	-149- (150) 150	-250- (250) 250	

- - denotes GRAVES variables

( ) denotes SUMT variables

no brackets are GRG variables

Problem 7

Source: Adapted from a model proposed by D.A. Schrady and U.C. Choe, [Ref. 13].

No. of independent variables: 100

No. of constraints: 1 linear constraint
1 nonlinear constraint
50 lower bounds on the variables

Objective function:

$$\text{minimize} \quad f(x) = \sum_{i=1}^{50} \frac{B_i(x_i + 50)}{x_i}$$

where

$$B_i(x_i+50) = \frac{1}{2}\left[s_i^2 + d_i^2\right] \Phi\left(\frac{d_i}{s_i}\right) - \frac{s_i d_i}{2} \phi\left(\frac{d_i}{s_i}\right)$$

and $$d_i = x_{(i+50)} - m_i$$

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

$$\Phi(r) = \int_r^x \phi(x)\, dx$$

Constraints:

$$K_1 - \sum_{i=1}^{50} c_i \left| \frac{x_i}{2} + x_{(i+50)} - m_i \right| \geq 0$$

$$K_2 - \sum_{i=1}^{50} \frac{L_i}{x_i} \geq 0$$

$$x_i \geq 0 \qquad i=1,\ldots,50$$

$$K_1 = 200{,}000$$

$$K_2 = 300$$

Data for Problem 7

	L_i's	c_i's	m_i's	s_i's
1	1000	1	100	100
2	1500	10	200	100
3	2000	20	300	200
4	1100	17	200	100
5	1900	23	100	100
6	700	8	200	200
7	400	12	200	200
8	1200	19	300	100
9	2000	2	500	200
10	1300	5	300	100
11	1900	21	100	100
12	900	16	200	200
13	1400	13	400	200
14	1500	19	500	300
15	2200	7	400	100
16	1700	4	300	100
17	1800	12	200	200
18	800	5	100	100
19	700	18	100	100
20	1100	16	100	100
21	1000	14	200	100
22	1800	21	200	200
23	1500	6	400	300
24	2100	6	500	100
25	1600	14	100	100

	l_i's	c_i's	m_i's	s_i's
26	700	2	100	100
27	2000	12	200	200
28	1800	3	500	300
29	1700	1	200	200
30	700	18	300	200
31	1200	19	100	100
32	1100	12	100	100
33	1700	9	500	100
34	600	8	300	100
35	400	1	200	100
36	1000	3	100	100
37	1900	17	400	300
38	1500	15	200	200
39	1400	18	400	300
40	1200	16	500	300
41	1300	5	100	100
42	1900	12	200	100
43	2000	15	300	200
44	2200	20	400	200
45	800	23	100	100
46	1900	17	200	200
47	2100	16	500	200
48	2000	4	500	300
49	500	8	100	100
50	900	12	100	100

Results for Problem 7

	initial	GRG	SUMT	GRAVES
f(x)	2008.2	80.89	80.73	193.03
x_1	300.0	319.18	441.75	358.57
x_2	300.0	201.55	202.13	283.48
x_3	300.0	235.40	235.52	350.71
x_4	300.0	154.48	153.15	195.25
x_5	300.0	171.36	168.97	139.96
x_6	300.0	235.21	213.68	199.96
x_7	300.0	189.50	181.01	181.23
x_8	300.0	154.23	152.95	233.91
x_9	300.0	472.64	479.21	307.91
x_{10}	300.0	249.19	247.75	230.58
x_{11}	300.0	175.54	173.38	136.72
x_{12}	300.0	203.45	203.56	224.15
x_{13}	300.0	239.65	232.42	201.47
x_{14}	300.0	294.19	286.03	285.43
x_{15}	300.0	266.53	269.92	179.99
x_{16}	300.0	289.56	304.16	257.52
x_{17}	300.0	252.24	253.34	344.82
x_{18}	300.0	208.60	203.41	266.16
x_{19}	300.0	132.43	131.50	89.60
x_{20}	300.0	157.06	155.69	236.01
x_{21}	300.0	157.71	156.65	172.25
x_{22}	300.0	228.17	228.01	244.51
x_{23}	300.0	344.97	335.74	328.76
x_{24}	300.0	300.04	281.65	326.07
x_{25}	300.0	186.27	184.75	282.89
x_{26}	300.0	271.19	277.62	164.98
x_{27}	300.0	259.50	261.33	240.90

	initial	GRG	SUMT	GRAVES
x_{28}	300.0	397.19	431.79	365.20
x_{29}	300.0	423.59	598.64	528.20
x_{30}	300.0	192.39	192.76	351.17
x_{31}	300.0	154.25	152.90	104.03
x_{32}	300.0	170.06	169.53	220.08
x_{33}	300.0	228.43	220.55	175.01
x_{34}	300.0	159.39	155.54	218.90
x_{35}	300.0	298.67	293,51	200.04
x_{36}	300.0	262.18	272.91	268.90
x_{37}	300.0	308.01	299.71	343.80
x_{38}	300.0	229.81	230.32	254.82
x_{39}	300.0	295.16	283.78	439.54
x_{40}	300.0	295.71	278.89	299.69
x_{41}	300.0	240.84	247.65	267.84
x_{42}	300.0	208.46	207.65	190.48
x_{43}	300.0	248.06	248.49	210.33
x_{44}	300.0	240.71	241.18	278.48
x_{45}	300.0	130.71	129.77	255.60
x_{46}	300.0	238.75	239.13	224.15
x_{47}	300.0	252.70	248.52	311.52
x_{48}	300.0	407.32	409.01	507.23
x_{49}	300.0	147.50	146.09	107.56
x_{50}	300.0	158.38	158.08	95.74
x_{51}	300.0	311.03	297.76	375.11
x_{52}	300.0	334.79	334.50	252.24
x_{53}	300.0	554.60	554.64	477.52
x_{54}	300.0	321.88	322.53	225.55
x_{55}	300.0	201.42	202.46	194.65
x_{56}	300.0	533.48	543.68	486.93

	initial	GRG	SUMT	GRAVES
x_{57}	300.0	517.88	523.81	498.57
x_{58}	300.0	416.59	417.16	368.46
x_{59}	300.0	893.45	889.23	811.32
x_{60}	300.0	455.31	456.33	495.29
x_{61}	300.0	205.00	205.83	246.11
x_{62}	300.0	488.69	488.64	409.38
x_{63}	300.0	690.71	695.32	572.72
x_{64}	300.0	916.04	918.54	765.45
x_{65}	300.0	538.04	537.61	588.42
x_{66}	300.0	459.55	457.11	524.78
x_{67}	300.0	494.65	494.78	399.30
x_{68}	300.0	264.98	264.70	199.96
x_{69}	300.0	226.60	227.10	233.91
x_{70}	300.0	223.96	224.66	236.01
x_{71}	300.0	330.03	330.70	286.07
x_{72}	300.0	452.63	453.08	403.50
x_{73}	300.0	940.70	944.58	816.66
x_{74}	300.0	639.98	642.67	672.35
x_{75}	300.0	222.06	222.85	184.75
x_{76}	300.0	290.09	289.03	370.86
x_{77}	300.0	492.25	492.03	392.38
x_{78}	300.0	1105.45	1095.59	918.73
x_{79}	300.0	654.97	624.10	556.15
x_{80}	300.0	582.95	582.95	551.76
x_{81}	300.0	216.58	217.18	250.88
x_{82}	300.0	233.56	234.21	220.08
x_{83}	300.0	633.98	635.33	589.73

	initial	GRG	SUMT	GRAVES
x_{84}	300.0	454.69	456.13	478.23
x_{85}	300.0	412.11	412.73	375.56
x_{86}	300.0	275.54	273.65	208.42
x_{87}	300.0	823.92	827.30	768.80
x_{88}	300.0	483.80	483.34	383.05
x_{89}	300.0	824.73	826.95	709.39
x_{90}	300.0	938.50	945.03	721.15
x_{91}	300.0	259.25	256.37	199.96
x_{92}	300.0	324.31	324.64	305.69
x_{93}	300.0	576.67	576.45	472.27
x_{94}	300.0	652.48	652.38	510.78
x_{95}	300.0	215.22	215.85	255.60
x_{96}	300.0	469.03	468.47	387.54
x_{97}	300.0	768.47	770.50	665.23
x_{98}	300.0	1069.70	1068.96	1096.86
x_{99}	300.0	258.91	258.82	215.21
x_{100}	300.0	236.84	237.43	220.08
$h_1(x)$	80650.0	1.7E-13	7.2E-01	7.9E-06
$h_2(x)$	68.0	2.4E-05	1.9E-04	3.6E-05
Time		96 sec	246 sec	234 sec

Alternate Initial Points:

a) $x_i = 10$ $\quad i=1,\ldots,100$

b) $x_i = 1000$ $\quad i=1,\ldots,100$

Problem 8

Source: Adapted from a model proposed by
A.J. Scott [Ref. 14].

No. of independent variables: 46

No. of constraints: 1 nonlinear inequality constraint
1 linear equality constraint
46 lower bounds on the variables

Objective function:

$$\text{Minimize} \quad f(x) = \sum_{i=1}^{46} \frac{x_i}{T} \left(\ln \frac{x_i}{T} \right)$$

Constraints:

$$T - \sum_{i=1}^{46} x_i = 0$$

$$S - \sum_{i=1}^{46} c_j y_i + a d_i^b y_i \geq 0$$

$$\text{where} \quad y_i = x_i + \sum_{j \in A(i)} x_j$$

A(i) consists of all arcs (in figure A1-1) that converge directly and indirectly upon node i.

$$x_i \geq 0 \qquad i=1,\ldots,46$$

Data for problem 8

$a = 0.05$ $\qquad$ $T = 500$

$b = 1.50$ $\qquad$ $S = 10000$

$c_1 = 5.0$	$c_2 = 4.0$	$c_3 = 5.0$
$c_4 = 7.0$	$c_5 = 7.0$	$c_6 = 8.0$
$c_7 = 6.0$	$c_8 = 4.0$	$c_9 = 3.0$
$c_{10} = 12.0$	$c_{11} = 14.0$	$c_{12} = 10.0$
$c_{13} = 21.0$	$c_{14} = 23.0$	$c_{15} = 8.0$
$c_{16} = 9.0$	$c_{17} = 10.0$	$c_{18} = 13.0$
$c_{19} = 20.0$	$c_{20} = 5.0$	$c_{21} = 8.0$
$c_{22} = 6.0$	$c_{23} = 3.0$	$c_{24} = 8.0$
$c_{25} = 11.0$	$c_{26} = 6.0$	$c_{27} = 18.0$
$c_{28} = 15.0$	$c_{29} = 10.0$	$c_{30} = 8.0$
$c_{31} = 8.0$	$c_{32} = 14.0$	$c_{33} = 11.0$
$c_{34} = 12.0$	$c_{35} = 14.0$	$c_{36} = 18.0$
$c_{37} = 16.0$	$c_{38} = 9.0$	$c_{39} = 2.0$
$c_{40} = 8.0$	$c_{41} = 11.0$	$c_{42} = 12.0$
$c_{43} = 11.0$	$c_{44} = 18.0$	$c_{45} = 20.0$
	$c_{46} = 13.0$	

$$d_i = c_i \qquad i=1,\ldots,46$$

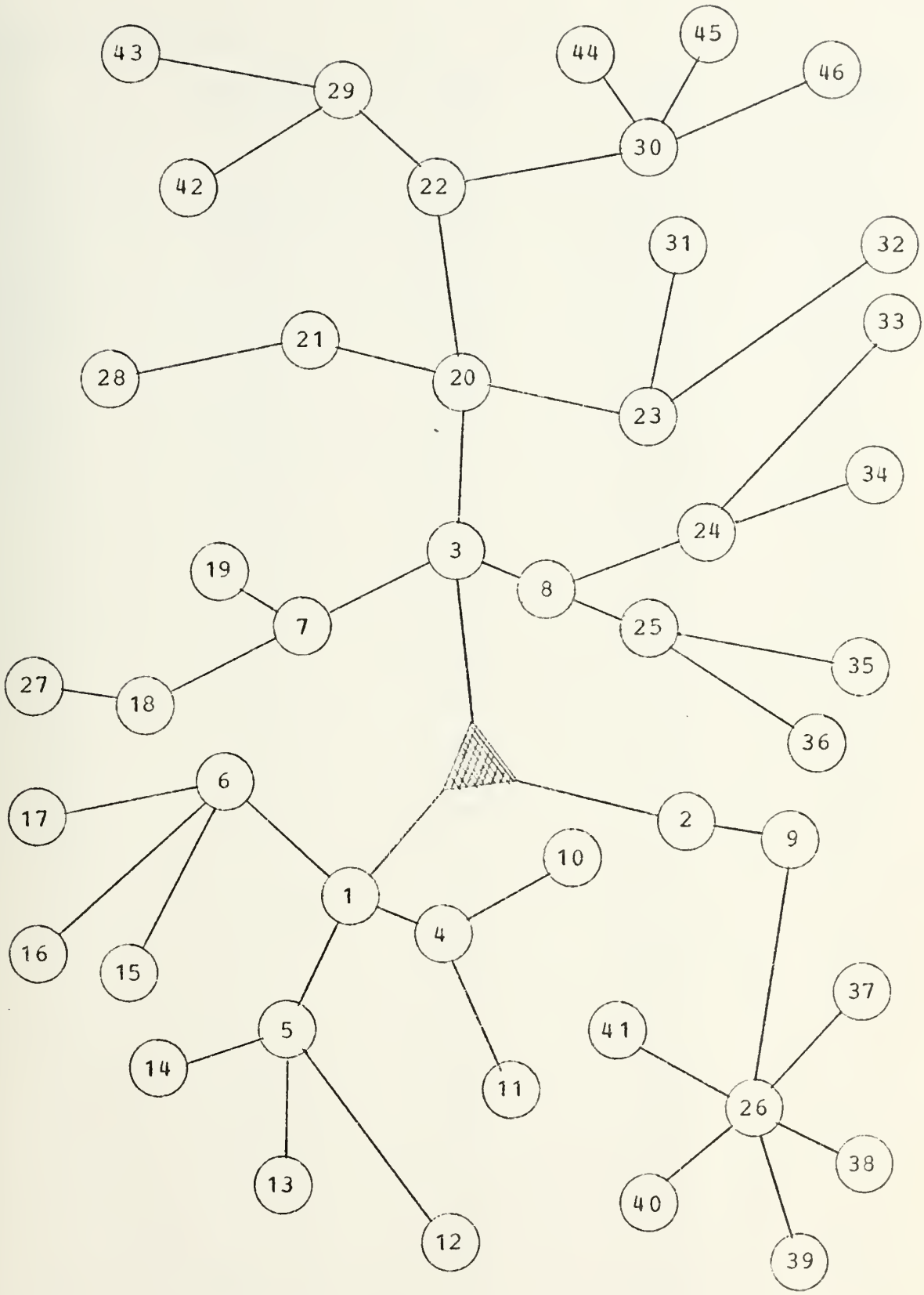

Figure A1-1. Transportation Routes for Problem 8

Results for Problem 8

	initial	GRG	SUMT	GRAVES
f(x)	-3.249	-3.401	-3.468	-3.462
x_1	40.000	34.86	39.08	37.14
x_2	26.667	35.66	45.24	44.36
x_3	86.667	72.80	36.46	40.81
x_4	10.000	13.22	19.36	18.99
x_5	13.333	13.99	19.25	19.32
x_6	13.333	13.35	17.02	17.73
x_7	13.333	13.53	19.97	20.65
x_8	23.333	19.37	23.45	22.16
x_9	23.333	22.06	31.56	30.48
x_{10}	3.333	8.48	7.17	6.76
x_{11}	3.333	7.58	6.18	6.03
x_{12}	3.333	9.26	8.30	8.79
x_{13}	3.333	4.49	3.75	3.39
x_{14}	3.333	3.72	3.28	3.03
x_{15}	3.333	9.60	8.67	8.20
x_{16}	3.333	9.13	8.02	8.19
x_{17}	3.333	8.67	7.43	6.87
x_{18}	6.667	5.86	6.70	6.38
x_{19}	3.333	4.56	4.15	4.07
x_{20}	46.667	34.68	19.96	21.98
x_{21}	6.667	7.99	9.86	8.99
x_{22}	26.667	9.46	11.06	12.56
x_{23}	10.000	10.62	14.91	14.16
x_{24}	10.000	9.02	11.14	11.25
x_{25}	10.000	7.42	8.82	8.44
x_{26}	20.000	16.12	16.33	16.71
x_{27}	3.333	1.00	1.85	1.57
x_{28}	3.333	2.96	3.17	3.24
x_{29}	10.000	3.29	4.83	4.71

	initial	GRG	SUMT	GRAVES
x_{30}	13.333	4.82	5.62	6.28
x_{31}	3.333	7.93	7.67	7.46
x_{32}	3.333	5.34	4.91	4.92
x_{33}	3.333	5.21	4.69	4.29
x_{34}	3.333	4.81	4.36	4.39
x_{35}	3.333	2.82	3.07	3.21
x_{36}	3.333	1.62	2.35	2.40
x_{37}	3.333	5.82	4.61	4.22
x_{38}	3.333	8.92	7.72	7.78
x_{39}	3.333	12.24	13.67	13.46
x_{40}	3.333	9.39	8.35	7.77
x_{41}	3.333	8.01	6.63	6.99
x_{42}	3.333	1.14	2.05	1.67
x_{43}	3.333	1.27	2.19	2.61
x_{44}	3.333	1.72	1.59	1.57
x_{45}	3.333	4.91	1.40	1.57
x_{46}	3.333	1.29	2.20	2.45
$h_1(x)$	80.330	-1.2E-05	8.9E-01	32.99
$h_2(x)$	-6.8E-13	-7.3E-15	6.2E-07	3.46
Time		32 sec	53.8 sec	150 sec

Alternate Initial Points:

a) $x_i = 10.87$ $\quad i=1,\ldots,46$

b) $x_i = 75.0$ $\quad i=1,2,3$

$x_i = 10.0$ $\quad i=4,\ldots,23$

$x_i = 3.26$ $\quad i=24,\ldots,46$

List of References

1. IBM Technical Report-320-2947, A Comparative Study on Nonlinear Programming Codes, by A.R. Colville, June 1968.

2. Fiacco, A.V. and McCormick, G.P., Nonlinear Programming Sequential Unconstrained Minimization Techniques, John Wiley and Sons, Inc., 1968.

3. Mylander, W.C., Holmes, R.L., and McCormick, G.P., A Guide to SUMT-Version 4: The Computer Program Implementing the Sequential Unconstrained Minimization Technique for Nonlinear Programming, Research Analysis Corporation, McLean, Va., February 1971.

4. Office of Naval Research Technical Memorandum CIS-75-01, GRG System Documentation, by L.S. Lasdon, A.D. Waren, M.W. Ratner, and A. Jain, November 1975.

5. Office of Naval Research Technical Memorandum CIS-75-02, GRG User's Guide, by L.S. Lasdon, A.D. Waren, M.W. Ratner, and A. Jain, November 1975.

6. Reklaitis, G.W. and Phillips, D.T., "A Survey of Nonlinear Programming," AIIE Transactions Vol. 7, No. 3, p 235-256 September, 1975.

7. Aerospace Corporation Report No. ATR-64(7040)-2, Development and Testing of a Nonlinear Programming Algorithm, by G.W. Graves, June 1964.

8. Winston, A.B., "The Application of a Nonlinear Algorithm to a Second-Order Representation of the Problem," Centre d'Etudes de Recherche Operationnelle, Vol. 11, p 75-91, 1969.

9. Graves, G.W., "A Complete Constructive Algorithm for the General Mixed Linear Programming Problem," Naval Research Logistics Quarterly, Vol. 12, No. 1, p 1-34 March, 1965.

10. Rand Corporation Report R-1411-DDPAE, Sortie Allocation by a Nonlinear Programming Model for Determining a Munitions Mix, by R.J. Clasen, G.W. Graves and J.Y. Lu, March 1974.

11. Himmelblau, D.M., Applied Nonlinear Programming, McGraw-Hill, 1972.

12. Bracken, J. and McCormick, G.P., Selected Applications of Nonlinear Programming, John Wiley and Sons, Inc., 1968.

13. Schrady, D.A. and Choe, U.C., "Models for Multi-item Continuous Review Inventory Policies Subject to Constraints," Naval Research Logistics Quarterly Vol. 18, No. 4, p 451-463 December, 1971.

14. Scott, A.J., "A Model of Nodal Entropy in a Transportation Network with Congestion Costs," Transportation Science Vol. 5, No. 2, p 204-211 May, 1971.

15. Research Analysis Corp. Report RAC-TP-272, The Chemical Equilibrium Problem: An Application of SUMT, by A.P. Jones, 1967.

16. Research Analysis Corp. Report RAC-TP-302, On Variable Metric Methods of Minimization, by J.D. Pearson, May, 1968.

17. Paviani, D.A. Ph.D. dissertation, University of Texas, Austin, Tex., 1969.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Professor G. H. Bradley (Thesis Advisor) Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	10
4.	Associate Professor G. G. Brown Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	1
5.	Lieutenant R. J. Waterman, U. S. Navy (Student) Hanover Road Forestville, New York 14062	1
6.	Chairman Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	1